OpenShift Container Platform 4.16

网络

配置和管理集群网络

Last Updated: 2024-07-02

OpenShift Container Platform 4.16 网络

配置和管理集群网络

法律通告

摘要

本文档提供有关配置和管理 OpenShift Container Platform 集群网络的说明，其中包括 DNS、Ingress 和 Pod 网络。

目录

第 1 章 关于网络

Red Hat OpenShift 网络是一个功能生态系统、插件和高级网络功能，它使用高级网络相关功能来扩展 Kubernetes 网络，集群需要为其一个或多个混合集群管理网络流量。这个网络功能生态系统集成了入口、出口、负载均衡、高性能吞吐量、安全性和集群内部流量管理，并提供基于角色的可观察工具来减少其自然复杂性。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

以下列表重点介绍集群中可用的一些最常用的 Red Hat OpenShift Networking 功能：

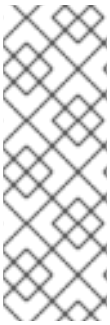
- 由以下 Container Network Interface (CNI) 插件之一提供的主要集群网络：
  - OVN-Kubernetes 网络插件，默认插件
  - OpenShift SDN 网络插件
- 经认证的第三方替代主网络插件
- 用于网络插件管理的 Cluster Network Operator
- 用于 TLS 加密 Web 流量的 Ingress Operator
- 用于名称分配的 DNS Operator
- 用于裸机集群上的流量负载均衡的 MetalLB Operator
- 对高可用性的 IP 故障转移支持
- 通过多个 CNI 插件支持额外的硬件网络，包括 macvlan、ipvlan 和 SR-IOV 硬件网络
- IPv4、IPv6 和双堆栈寻址
- 用于基于 Windows 的工作负载的混合 Linux-Windows 主机集群
- Red Hat OpenShift Service Mesh 用于发现、负载均衡、服务对服务身份验证、故障恢复、指标和监控服务
- 单节点 OpenShift
- Network Observability Operator 用于网络调试和见解
- Submariner 用于 inter-cluster 网络
- Red Hat Service Interconnect 用于第 7 层 inter-cluster 网络

第 2 章 了解网络

集群管理员有几个选项用于公开集群内的应用程序到外部流量并确保网络连接：

- 服务类型，如节点端口或负载均衡器
- API 资源，如 **Ingress** 和 **Route**

默认情况下，Kubernetes 为 pod 内运行的应用分配内部 IP 地址。Pod 及其容器可以网络，但集群外的客户端无法访问网络。当您将应用公开给外部流量时，为每个容器集指定自己的 IP 地址意味着 pod 在端口分配、网络、命名、服务发现、负载平衡、应用配置和迁移方面可被视为物理主机或虚拟机。



注意

一些云平台提供侦听 169.254.169.254 IP 地址的元数据 API，它是 IPv4 **169.254.0.0/16** CIDR 块中的 连接内部 IP 地址。

此 CIDR 块无法从 pod 网络访问。需要访问这些 IP 地址的 Pod 必须通过将 pod spec 中的 **spec.hostnetwork** 字段设置为 **true** 来获得主机网络访问。

如果允许 pod 主机网络访问，则将授予 pod 对底层网络基础架构的访问权限。

2.1. OPENSHIFT CONTAINER PLATFORM DNS

如果您运行多个服务，比如使用多个 pod 的前端和后端服务，则要为用户名和服务 IP 等创建环境变量，使前端 pod 可以跟后端服务通信。如果删除并重新创建服务，可以为该服务分配一个新的 IP 地址，而且需要重新创建前端 pod 来获取服务 IP 环境变量的更新值。另外，必须在任何前端 pod 之前创建后端服务，以确保正确生成服务 IP，并将它作为环境变量提供给前端 pod。

因此，OpenShift Container Platform 具有一个内置 DNS，以便服务 DNS 以及服务 IP/端口能够访问这些服务。

2.2. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR

在创建 OpenShift Container Platform 集群时，在集群中运行的 Pod 和服务会各自分配自己的 IP 地址。IP 地址可供附近运行的其他容器集和服务访问，但外部客户端无法访问这些 IP 地址。Ingress Operator 实现 **IngressController** API，是负责启用对 OpenShift Container Platform 集群服务的外部访问的组件。

Ingress Operator 通过部署和管理一个或多个基于 HAProxy 的 Ingress Controller 来处理路由，使外部客户端可以访问您的服务。您可以通过指定 OpenShift Container Platform **Route** 和 Kubernetes **Ingress** 资源，来使用 Ingress Operator 路由流量。Ingress Controller 中的配置（如定义 **endpointPublishingStrategy** 类型和内部负载平衡）提供了发布 Ingress Controller 端点的方法。

2.2.1. 路由和 Ingress 的比较

OpenShift Container Platform 中的 Kubernetes Ingress 资源通过作为集群内 pod 运行的共享路由器服务来实现 Ingress Controller。管理 Ingress 流量的最常见方法是使用 Ingress Controller。您可以像任何其他常规 pod 一样扩展和复制此 pod。此路由器服务基于 HAProxy，后者是一个开源负载均衡器解决方案。

OpenShift Container Platform 路由为集群中的服务提供入口流量。路由提供了标准 Kubernetes Ingress Controller 可能不支持的高级功能，如 TLS 重新加密、TLS 直通和为蓝绿部署分割流量。

入口流量通过路由访问集群中的服务。路由和入口是处理入口流量的主要资源。Ingress 提供类似于路由的功能，如接受外部请求并根据路由委派它们。但是，对于 Ingress，您只能允许某些类型的连接：

HTTP/2、HTTPS 和服务器名称识别(SNI)，以及 TLS（证书）。在 OpenShift Container Platform 中，生成路由以满足 Ingress 资源指定的条件。

2.3. OPENSHIFT CONTAINER PLATFORM 网络的常见术语表

该术语表定义了在网络内容中使用的常用术语。

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。

AWS Load Balancer Operator

AWS Load Balancer (ALB) Operator 部署和管理 **aws-load-balancer-controller** 的实例。

Cluster Network Operator

Cluster Network Operator(CNO)在 OpenShift Container Platform 集群中部署和管理集群网络组件。这包括在安装过程中为集群选择的 Container Network Interface (CNI) 网络插件部署。

配置映射

配置映射提供将配置数据注入 pod 的方法。您可以在类型为 **ConfigMap** 的卷中引用存储在配置映射中的数据。在 pod 中运行的应用程序可以使用这个数据。

自定义资源 (CR)

CR 是 Kubernetes API 的扩展。您可以创建自定义资源。

DNS

集群 DNS 是一个 DNS 服务器，它为 Kubernetes 服务提供 DNS 记录。由 Kubernetes 启动的容器会在其 DNS 搜索中自动包含此 DNS 服务器。

DNS Operator

DNS Operator 部署并管理 CoreDNS，以便为 pod 提供名称解析服务。这会在 OpenShift Container Platform 中启用基于 DNS 的 Kubernetes 服务发现。

部署

维护应用程序生命周期的 Kubernetes 资源对象。

domain

Domain（域）是 Ingress Controller 提供的 DNS 名称。

egress

通过来自 pod 的网络出站流量进行外部数据共享的过程。

外部 DNS Operator

External DNS Operator 部署并管理 ExternalDNS，以便为从外部 DNS 供应商到 OpenShift Container Platform 的服务和路由提供名称解析。

基于 HTTP 的路由

基于 HTTP 的路由是一个不受保护的路由，它使用基本的 HTTP 路由协议，并在未安全的应用程序端口上公开服务。

入口

OpenShift Container Platform 中的 Kubernetes Ingress 资源通过作为集群内 pod 运行的共享路由器服务来实现 Ingress Controller。

Ingress Controller

Ingress Operator 管理 Ingress Controller。使用 Ingress Controller 是允许从外部访问 OpenShift Container Platform 集群的最常用方法。

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 NMState Operator

Kubernetes NMState Operator 提供了一个 Kubernetes API，用于使用 NMState 在 OpenShift Container Platform 集群的节点上执行状态驱动的网络配置。

kube-proxy

kube-proxy 是一个代理服务，在每个节点上运行，有助于为外部主机提供服务。它有助于将请求转发到正确的容器，并且能够执行原语负载平衡。

负载均衡器

OpenShift Container Platform 使用负载均衡器从集群外部与集群中运行的服务进行通信。

MetalLB Operator

作为集群管理员，您可以将 MetalLB Operator 添加到集群中，以便在将 **LoadBalancer** 类型服务添加到集群中时，MetalLB 可为该服务添加外部 IP 地址。

multicast

通过使用 IP 多播，数据可同时广播到许多 IP 地址。

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 Container Platform Ingress Operator

Ingress Operator 实现 **IngressController** API，是负责启用对 OpenShift Container Platform 服务的外部访问的组件。

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。

PTP Operator

PTP Operator 会创建和管理 **linuxptp** 服务。

route

OpenShift Container Platform 路由为集群中的服务提供入口流量。路由提供了标准 Kubernetes Ingress Controller 可能不支持的高级功能，如 TLS 重新加密、TLS 直通和为蓝绿部署分割流量。

扩展

增加或减少资源容量。

service

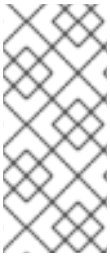
在一组 pod 上公开正在运行的应用程序。

单根 I/O 虚拟化 (SR-IOV) Network Operator

Single Root I/O Virtualization（SR-IOV）Network Operator 管理集群中的 SR-IOV 网络设备和网络附加。

软件定义型网络 (SDN)

OpenShift Container Platform 使用软件定义网络 (SDN) 方法来提供一个统一的集群网络，它允许 OpenShift Container Platform 集群中的不同 pod 相互间进行通信。



注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

流控制传输协议 (SCTP)

SCTP 是基于信息的可靠协议，可在 IP 网络之上运行。

taint

污点和容限可确保将 pod 调度到适当的节点上。您可以在节点上应用一个或多个污点。

容限 (tolerations)

您可以将容限应用到 pod。容限 (toleration) 允许调度程序调度具有匹配污点的 pod。

Web 控制台

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

第 3 章 零信任网络

零信任（Zero trust）一个设计安全基础架构的方法，它基于内部的环境，其中的每个交流行为都从一个不受信任的状态开始。这与传统的架构不同。传统架构可能会根据交流是否是在防火墙内发起的来确定其信任性。更具体地说，零信任会尝试缩小安全基础架构中的漏洞（依赖隐式信任模型和一次性身份验证）。

OpenShift Container Platform 可以为平台上运行的容器添加一些零信任网络功能，而无需更改容器或它们中运行的软件。红帽提供的几种产品进一步增加了容器的零信任网络功能。如果您能够更改容器中运行的软件，则还有红帽支持的其他项目，可以添加进一步的功能。

探索零信任网络的以下目标功能。

3.1. 信任的根

公共证书和私钥对于零信任网络至关重要。它们用来识别另一个组件、验证和保护流量的安全组件。证书由其他证书签名，存在到根证书颁发机构 (CA) 的信任链。参与网络的所有内容需要最终具有 root CA 的公钥，以便它可以验证信任链。对于面向公共的项，通常是全局已知的根 CA，其密钥与操作系统、Web 浏览器等一起分发。但是，如果私有 CA 的证书被分发到所有方，则可以针对一个集群或一个机构运行私有 CA。

利用：

- OpenShift Container Platform：OpenShift 在安装时会创建一个集群 CA，用于保护集群资源。但是，OpenShift Container Platform 也可以为 集群中的服务创建并签署证书，并可在请求时将集群 CA 捆绑包注入 pod。OpenShift Container Platform 创建并 签名的证书 有 26 个月的生存时间(TTL)，并在 13 个月自动轮转。如有必要，也可以手动轮转它们。

- OpenShift cert-manager Operator: cert-manager 允许您请求由外部信任根签名的证书的密钥。有许多可配置的签发者可以与外部签发者集成，以及与委派的签名证书一起运行的方法。cert-manager API 可供零信任网络中的其他软件使用，以请求必要的证书（如 Red Hat OpenShift Service Mesh），或者可由客户软件直接使用。

3.2. 流量身份验证和加密

确保网络上的所有流量都加密，并且端点是可识别的。例如，Mutual TLS 或 mTLS，这是 mutual 身份验证的方法。

利用：

- OpenShift Container Platform：使用透明 pod-to-pod IPsec，可以通过 IP 地址识别流量的源和目标。这可以实现对出口流量使用 IPsec 加密。通过使用 egress IP 功能，流量的源 IP 地址可用于识别集群中的流量源。

- Red Hat OpenShift Service Mesh ：提供强大的 mTLS 功能，可透明地增强离开 pod 的流量以提供身份验证和加密。

- OpenShift cert-manager Operator ：使用自定义资源定义(CRD)请求可为您的程序挂载的证书，以用于 SSL/TLS 协议。

3.3. 身份识别和验证

在使用 CA 最小化证书后，您可以通过验证连接端的另一端的连接身份验证一个用户或客户端机器来建立信任关系。这还需要管理证书生命周期，以便在被破坏时限制使用。

利用：

- OpenShift Container Platform： 集群签名的服务证书，以确保客户端与可信端点通信。这要求服务使用 SSL/TLS，并且客户端使用 集群 CA。客户端身份必须使用某种其他方法提供。
- Red Hat Single Sign-On ：提供与企业用户目录或第三方身份提供程序进行身份验证集成。
- Red Hat OpenShift Service Mesh ：透明地升级到 mTLS、自动轮转、自定义证书过期和请求使用 JSON Web 令牌(JWT)的身份验证。
- OpenShift cert-manager Operator ：创建和管理供您的应用程序使用的证书。证书可以由 CRD 控制并挂载为 secret，也可以更改应用程序以直接与 cert-manager API 交互。

3.4. 服务间的授权

务必要根据请求者的身份控制对服务的访问。这由平台完成，不需要每个应用程序来实现它。这样可以更好地审核和检查策略。

利用：

- OpenShift Container Platform：可以使用 Kubernetes **NetworkPolicy** 和 **AdminNetworkPolicy** 对象在平台的网络层中强制实施隔离。
- Red Hat OpenShift Service Mesh ：使用标准 Istio 对象 进行 L4 和 L7 控制流量，并使用 mTLS 来识别流量的源和目的地，然后根据该信息应用策略。

3.5. 事务级验证

除了识别和验证连接的功能外，控制单个事务的访问也很有用。这可包括源、可观察性和语义验证的速率限制。

利用：

- Red Hat OpenShift Service Mesh ：执行 L7 检查请求，拒绝不正确的 HTTP 请求、事务级 可观察性和报告。Service Mesh 也可以使用 JWT 提供基于请求的身份验证。

3.6. 风险评估

随着集群中的安全策略数量增加，策略允许和拒绝的视觉化变得越来越重要。这些工具可让您更轻松地创建、视觉化和管理集群安全策略。

利用：

- Red Hat OpenShift Service Mesh ：使用 OpenShift Web 控制台创建和视觉化 Kubernetes **NetworkPolicy** 和 **AdminNetworkPolicy** 对象，以及 OpenShift Networking **EgressFirewall** 对象。
- Red Hat Advanced Cluster Security for Kubernetes ：对象的高级视觉化。

3.7. 站点范围内的策略实施和分发

在集群中部署应用程序后，管理组成安全规则的所有对象就变得困难。务必要应用站点范围的策略，并审核部署的对象是否符合策略。这应该允许将某些权限委派给定义的绑定内的用户和集群管理员，并应允许例外策略。

利用：

- Red Hat OpenShift Service Mesh：RBAC 来控制策略对象和委派控制。
- Red Hat Advanced Cluster Security for Kubernetes ：策略实施 引擎。
- Red Hat Advanced Cluster Management (RHACM) for Kubernetes：集中式策略控制。

3.8. 持续的可观察性，以及重新检查评估

运行的集群后，您希望能够观察流量，并使用定义的规则验证流量是否被端口。这对入侵检测、共识非常重要，有助于操作负载管理。

利用：

- Network Observability Operator ：允许在与集群中的 pod 和节点的网络连接时检查、监控和警报。
- Red Hat Advanced Cluster Management (RHACM) for Kubernetes：监控、收集和评估系统级事件，如进程执行、网络连接和流以及特权升级。它可以决定集群的基线，然后检测异常活动并提醒您。
- Red Hat OpenShift Service Mesh ： 可以监控进入和离开 pod 的流量。
- Red Hat OpenShift distributed tracing Platform ：对于适用检测应用程序，您可以看到与特定操作关联的所有流量，因为它划分为子请求到微服务。这可让您识别分布式应用程序中的瓶颈。

3.9. 端点安全性

务必要信任在集群中运行服务的软件没有被破坏。例如，您可能需要确保在可信硬件上运行认证的镜像，并有策略只允许基于端点特征或来自端点特征的连接。

利用：

- OpenShift Container Platform：Secureboot 可以确保集群中的节点正在运行可信软件，因此平台本身（包括容器运行时）没有被篡改。您可以将 OpenShift Container Platform 配置为仅 运行由某些签名签名的镜像。
- Red Hat Trusted Artifact Signer ：这可用于可信构建链并生成已签名的容器镜像。

3.10. 在集群外扩展信任

您可能希望通过允许集群到子域的 mint CA 来在集群外扩展信任。或者，您可能希望 attest 到集群中的工作负载身份到远程端点。

利用：

- OpenShift cert-manager Operator ：您可以使用 cert-manager 管理委托的 CA，以便您可以在不同的集群或机构间分发信任。
- Red Hat OpenShift Service Mesh ：可以使用 SPIFFE 向远程或本地集群中运行的端点提供远程测试工作负载。

第 4 章 访问主机

了解如何创建堡垒主机来访问 OpenShift Container Platform 实例，以及使用安全 shell (SSH) 访问 control plane 节点。

4.1. 访问安装程序置备的基础架构集群中 AMAZON WEB SERVICES 上的主机

OpenShift Container Platform 安装程序不会为任何置备 OpenShift Container Platform 集群的 Amazon Elastic Compute Cloud (Amazon EC2) 实例创建公共 IP 地址。为了可以 SSH 到 OpenShift Container Platform 主机，您必须按照以下步骤操作。

流程

1. 创建一个安全组，允许 SSH 访问由 **openshift-install** 命令创建的虚拟私有云 (VPC) 。
2. 在安装程序创建的某个公共子网中创建 Amazon EC2 实例。
3. 将公共 IP 地址与您创建的 Amazon EC2 实例相关联。
   与 OpenShift Container Platform 安装不同，您应该将您创建的 Amazon EC2 实例与 SSH 密钥对关联。这与您为这个实例选择的操作系统无关，因为它只是一个 SSH 堡垒将互联网桥接到 OpenShift Container Platform 集群的 VPC。它与您使用的 Amazon Machine Image (AMI) 相关。例如，在 Red Hat Enterprise Linux CoreOS（RHCOS） 中，您可以像安装程序一样通过 Ignition 提供密钥。
4. 一旦置备了 Amazon EC2 实例并可以 SSH 到它，您必须添加与 OpenShift Container Platform 安装关联的 SSH 密钥。这个密钥可以与堡垒实例的密钥不同，也可以相同。

注意

直接通过 SSH 访问仅建议在灾难恢复时使用。当 Kubernetes API 正常工作时，应该使用特权 Pod。

5. 运行 **oc get nodes**，查看输出结果，然后选择一个 master 节点。主机名类似于 **ip-10-0-1-163.ec2.internal**。
6. 从您手动部署到 Amazon EC2 的堡垒 SSH 主机中，SSH 部署到该 control plane 主机。确定您使用了在安装过程中指定的相同的 SSH 密钥：

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

第 5 章 网络仪表板

网络指标可在 OpenShift Container Platform web 控制台的 dashboard 中的 **Observe → Dashboards** 下查看。

5.1. NETWORK OBSERVABILITY OPERATOR

如果安装了 Network Observability Operator，可以通过从 **Dashboards** 下拉列表中选择 **Netobserv** 仪表板来查看网络流量指标仪表板。有关此仪表板中可用指标的更多信息，请参阅 Network Observability metrics dashboard。

5.2. 网络和 OVN-KUBERNETES 仪表板

您可以从仪表板中查看常规网络指标和 OVN-Kubernetes 指标。

要查看常规网络指标，请从 Dashboards 下拉列表中选择 Networking/Linux Subsystem Stats。您可以从控制面板查看以下网络指标：Network Utilisation、Network Saturation 和 Network Errors。

要查看 OVN-Kubernetes 指标，请从 Dashboards 下拉列表中选择 Networking/Infrastructure。您可以查看以下 OVN-Kuberenetes 指标：Networking Configuration, TCP Latency Probes, Control Plane Resources, 和 Worker Resources。

5.3. INGRESS OPERATOR 仪表板

您可以从仪表板中查看 Ingress Operator 处理的网络指标。这包括类似如下的指标：

- **传入和传出带宽**
- **HTTP 错误率**
- **HTTP 服务器响应延迟**

要查看这些 Ingress 指标，请从 Dashboards 下拉列表中选择 Networking/Ingress。您可以查看以下类别的 Ingress 指标：Top 10 Per Route, Top 10 Per Namespace, 和 Top 10 Per Shard。

第 6 章 网络安全性

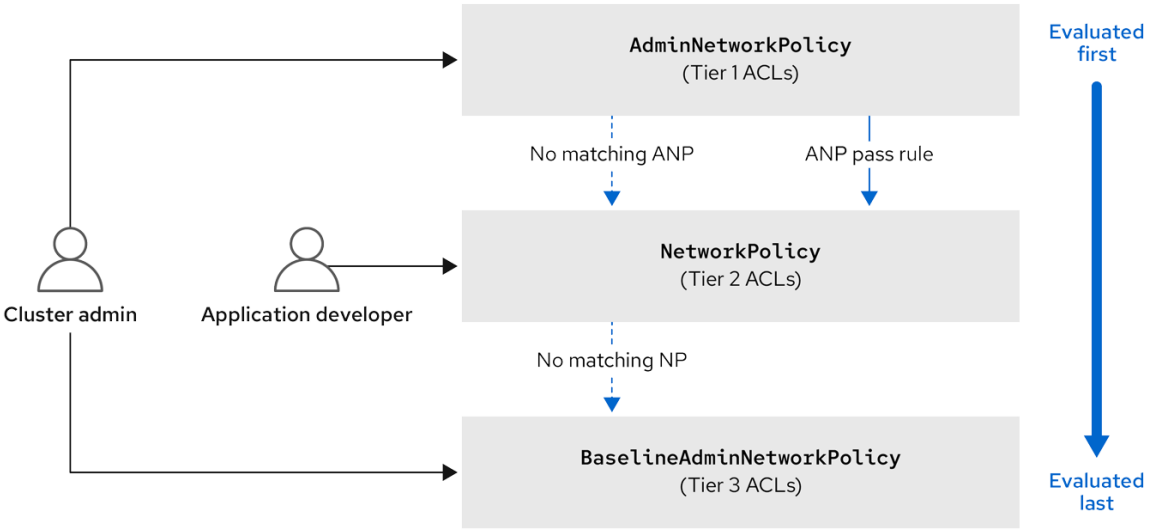
6.1. 了解网络策略 API

Kubernetes 提供了两个用户可用于强制实施网络安全的功能。允许用户强制执行网络策略的一个功能是 NetworkPolicy API，主要用于应用程序开发人员和命名空间租户，通过创建命名空间范围的策略来保护其命名空间。

第二个功能是 AdminNetworkPolicy，它由两个 API 组成：AdminNetworkPolicy (ANP) API 和 BaselineAdminNetworkPolicy (BANP) API。ANP 和 BANP 是为集群和网络管理员设计的，以通过创建集群范围的策略来保护其整个集群。集群管理员可以使用 ANPs 来强制实施优先于 NetworkPolicy 对象的不可覆盖的策略。管理员可以使用 BANP 设置并强制实施可选的集群范围的网络策略规则，当需要时，用户可以使用 NetworkPolicy 对象覆盖它。当一起使用时，ANP、BANP 和网络策略可以实现完整的多租户隔离，管理员可使用这个功能保护集群。

OpenShift Container Platform 中的 OVN-Kubernetes CNI 使用访问控制列表(ACL) Tiers 实施这些网络策略，以评估并应用它们。ACL 按照从 Tier 1 到 Tier 3 的降序进行评估。

第 1 级评估 AdminNetworkPolicy (ANP)对象。第 2 级评估 NetworkPolicy 对象。第 3 级评估 BaselineAdminNetworkPolicy (BANP)对象。



615_OpenShift_0324

首先评估 ANP。当匹配是 ANP allow 或 deny 规则时，集群中的任何现有 NetworkPolicy 和 BaselineAdminNetworkPolicy（BANP）对象将不会被评估。当匹配是 ANP pass，评估会从 ACL 的第 1 层移到第 2 层，在其中评估 NetworkPolicy 策略。如果没有 NetworkPolicy 与流量匹配，则评估从第 2 层 ACL 移到评估 BANP 的第 3 层 ACL。

6.1.1. AdminNetworkPolicy 和 NetworkPolicy 自定义资源之间的主要区别

下表解释了集群范围的 AdminNetworkPolicy API 和命名空间范围 NetworkPolicy API 之间的主要区别。

策略元素	AdminNetworkPolicy	NetworkPolicy
适用的用户	集群管理员	命名空间所有者

策略元素	AdminNetworkPolicy	NetworkPolicy
影响范围	Cluster	Namespaced
丢弃流量	当显式 **Deny** 操作设置为一个规则时支持。	在策略创建时通过隐式 **Deny** 隔离时支持。
委派流量	**Pass** 操作设置为一个规则时被支持。	Not applicable
允许流量	显式 **Allow** 操作设置一个规则时被支持。	所有规则的默认操作都是 allow。
策略中的规则优先级	取决于它们出现在 ANP 中的顺序。位置更高的规则有更高的优先级。	规则的效果是叠加的
策略优先级	在 ANP 中，**priority** 字段用于设置评估的顺序。策略优先级号越低，优先级越高。	在策略之间没有策略排序。
功能优先级	首先通过 1 层 ACL 和 BANP 评估，最后通过第 3 层 ACL 评估。	在 ANP 之后，BANP 之前强制实施，它们会在 ACL 层 2 中进行评估。
匹配 pod 选择	在命名空间之间可应用不同的规则。	可以在单一命名空间中的 pod 之间应用不同的规则。
集群出口流量	通过**节点**和**网络** 对等点支持	通过 **ipBlock** 字段以及接受的 CIDR 语法支持。
集群入口流量	不支持	不支持
完全限定域名 (FQDN) 对等支持	不支持	不支持
命名空间选择器	支持通过使用 **namespaces.matchLabels** 字段进行命名空间的高级选择	支持使用 **namespaceSelector** 字段支持基于标签的命名空间选择

6.2. 管理网络策略

6.2.1. OVN-Kubernetes AdminNetworkPolicy

6.2.1.1. AdminNetworkPolicy

AdminNetworkPolicy (ANP)是一个集群范围的自定义资源定义(CRD)。作为 OpenShift Container Platform 管理员，您可以在创建命名空间前通过创建网络策略来使用 ANP 来保护网络。另外，您可以在集群范围的级别上创建网络策略，该级别不可由 NetworkPolicy 对象覆盖。

AdminNetworkPolicy 和 NetworkPolicy 对象之间的关键区别在于，供管理员使用，是集群范围，而后者则用于租户所有者，并且是命名空间范围。

ANP 允许管理员指定以下内容：

- 确定其评估顺序的 priority 值。数值越低，优先级越高。
- 由应用策略的一组命名空间或命名空间组成的一组 pod。
- 要应用到 subject 的所有入口流量的入站规则列表。
- 用于来自 subject 的所有出口流量的出口规则列表。

AdminNetworkPolicy 示例

例 6.1. ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: sample-anp-deny-pass-rules (1)
spec:
  priority: 50 (2)
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name (3)
  ingress: (4)
  - name: "deny-all-ingress-tenant-1" (5)
    action: "Deny"
    from:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-anp: tenant-1
        podSelector:
          matchLabels:
            custom-anp: tenant-1 (6)
  egress:(7)
  - name: "pass-all-egress-to-tenant-1"
    action: "Pass"
    to:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-anp: tenant-1
        podSelector:
          matchLabels:
            custom-anp: tenant-1
```

(1) 为您的 ANP 指定一个名称。

(2) spec.priority 字段支持集群中 0-99 值中最多 100 ANP。数值越低，优先级越高。创建具有相同优先级的 AdminNetworkPolicy 会创建非确定的结果。

(3) 指定要应用 ANP 资源的命名空间。

4 ANP 具有入口和出口规则。spec.ingress 字段的 ANP 规则接受Pass,Deny, action 字段接受的值为 Allow。

5 为 ingress.name 指定一个名称。

6 指定 podSelector.matchLabels，以选择 namespaceSelector.matchLabels 作为入口对等选择的命名空间中的 pod。

7 ANP 同时具有入口和出口规则。spec.egress 字段的 ANP 规则接受Pass,Deny, action 字段接受的值为 Allow。

其他资源

- Network Policy API 工作组

6.2.1.1.1. 规则的 AdminNetworkPolicy 操作

作为管理员，您可以将您的 AdminNetworkPolicy 规则的 action 字段设置为 Allow,Deny, 或 Pass。由于 OVN-Kubernetes 使用分层 ACL 来评估网络流量规则，因此 3NP 允许您设置非常强大的策略规则，它们只能被管理员修改、删除规则，或通过设置更高优先级规则来覆盖它们。

AdminNetworkPolicy Allow 示例
在优先级 9 中定义的以下 ANP 可确保允许从 monitoring 命名空间到集群中的任何租户（所有其他命名空间）的所有入口流量。

例 6.2. 强 Allow ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: allow-monitoring
spec:
  priority: 9
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift namespaces as well.
  ingress:
  - name: "allow-ingress-from-monitoring"
    action: "Allow"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这是强的 Allow ANP 的示例，因为它不可以被涉及的所有方覆盖。租户都不会阻止自己被使用 NetworkPolicy 对象监控，监控租户也不知道它可以或无法监控的内容。

AdminNetworkPolicy 拒绝示例
在优先级 5 中定义的以下 ANP 可确保 monitoring 命名空间中的所有入口流量都被阻止到受限租户（具有标签 security: restricted的命名空间）。

例 6.3. 强 Deny ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: block-monitoring
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        security: restricted
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这是一个强大的 Deny ANP，这是所有涉及的方都无法覆盖的。受限租户所有者无法授权自己允许监控流量，基础架构监控服务无法从这些敏感命名空间中提取任何内容。

与强的 Allow 示例结合使用时，block-monitoring ANP 具有较低优先级的值，赋予其优先级更高的优先级，这样可确保不会监控受限租户。

AdminNetworkPolicy Pass 示例
在优先级 7 定义的以下 ANP 可确保所有从 monitoring 命名空间到内部基础架构租户（具有标签 security: internal）的入口流量都将传递到 ACL 的层 2，并由命名空间的NetworkPolicy 对象评估。

例 6.4. 强 Pass ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: pass-monitoring
spec:
  priority: 7
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "pass-ingress-from-monitoring"
    action: "Pass"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这个示例是一个强大的 Pass 操作 ANP，因为它将决策委派给租户所有者定义的 NetworkPolicy 对象。如果基础架构监控服务应使用命名空间范围 NetworkPolicy 对象提取其指标，则此 pass-monitoring ANP 允许在安全级别 internal 分组的所有租户所有者。

6.2.2. OVN-Kubernetes BaselineAdminNetworkPolicy

6.2.2.1. BaselineAdminNetworkPolicy

BaselineAdminNetworkPolicy (BANP)是一个集群范围的自定义资源定义(CRD)。作为 OpenShift Container Platform 管理员，您可以使用 BANP 来设置并强制实施可选的基准网络策略规则，这些规则被用户使用 NetworkPolicy 对象（如果需要的话）覆盖。BANP 的规则操作是 allow 或 deny。

BaselineAdminNetworkPolicy 资源是一个集群单例对象，当传递的流量策略与集群中的任何 NetworkPolicy 对象不匹配时，可用作 guardrail 策略。BANP 也可以用作默认安全模型，该模型默认阻止集群内流量，用户需要使用 NetworkPolicy 对象来允许已知的流量。在创建 BANP 资源时，必须使用 default 作为名称。

管理员可通过 BANP 指定：

- 由一组命名空间或命名空间的 subject。
- 要应用到 subject 的所有入口流量的入站规则列表。
- 用于来自 subject 的所有出口流量的出口规则列表。

BaselineAdminNetworkPolicy 示例

例 6.5. BANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default (1)
spec:
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name (2)
  ingress: (3)
  - name: "deny-all-ingress-from-tenant-1" (4)
    action: "Deny"
    from:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-banp: tenant-1 (5)
        podSelector:
          matchLabels:
            custom-banp: tenant-1 (6)
  egress:
  - name: "allow-all-egress-to-tenant-1"
    action: "Allow"
    to:
    - pods:
        namespaceSelector:
```

```
      matchLabels:
        custom-banp: tenant-1
    podSelector:
      matchLabels:
        custom-banp: tenant-1
```

1. 策略名称必须是 default，因为 BANP 是一个单例对象。
2. 指定要将 ANP 应用到的命名空间。
3. BANP 具有入口和出口规则。spec.ingress 和 spec.egress 字段的 BANP 规则接受 Deny, action 字段接受的值为 Allow。
4. 为 ingress.name 指定名称
5. 指定要从中选择 pod 以应用 BANP 资源的命名空间。
6. 指定 podSelector.matchLabels 名称，以应用 BANP 资源。

BaselineAdminNetworkPolicy 拒绝示例

以下 BANP 单例确保管理员为 internal 安全级别进入租户的所有入口监控流量设置了默认的拒绝策略。与 "AdminNetworkPolicy Pass example" 组合时，这个 deny 策略充当 ANP pass-monitoring 策略传递的所有入口流量的保护策略。

例 6.6. guardrail Deny 规则的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

您可以将带有 action 字段的值为 Pass 的 AdminNetworkPolicy 资源与 BaselineAdminNetworkPolicy 资源结合使用来创建多租户策略。此多租户策略允许一个租户在应用上收集监控数据，同时不从第二个租户收集数据。

作为管理员，如果您同时应用了 "AdminNetworkPolicy Pass action example" 和 "BaselineAdminNetwork Policy Deny example"，则租户将保留创建在 BANP 之前评估的 NetworkPolicy 资源。

例如，租户 1 可以设置以下 NetworkPolicy 资源来监控入口流量：

例 6.7. NetworkPolicy 示例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-monitoring
  namespace: tenant 1
spec:
  podSelector:
  policyTypes:
    - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

在这种情况下，Tenant 1 会在 "AdminNetworkPolicy Pass action example" 之后，"BaselineAdminNetwork Policy Deny example" 之前被评估，它将拒绝所有进入安全级别 internal 的入口监控流量。随着租户 1 的 NetworkPolicy 对象就位，它们将能够在其应用程序中收集数据。但是，租户 2 没有任何 NetworkPolicy 对象，将无法收集数据。作为管理员，您没有默认监控内部租户，而是创建了 BANP，它允许租户使用 NetworkPolicy 对象覆盖 BANP 的默认行为。

6.2.3. 监控 ANP 和 BANP

AdminNetworkPolicy 和 BaselineAdminNetworkPolicy 资源具有可用于监控和管理您的策略的指标。有关指标的详情，请查看下表。

6.2.3.1. AdminNetworkPolicy 指标

Name	描述	解释
ovnkube_controller_admin_network_policies	Not applicable	集群中的 **AdminNetworkPolicy** 资源总数。
ovnkube_controller_baseline_admin_network_policies	Not applicable	集群中的 **BaselineAdminNetworkPolicy** 资源总数。该值应该是 0 或 1。
ovnkube_controller_admin_network_policies_rules	• **direction**: 指定 **Ingress** 或 **Egress**。 • **action**: 指定 **Pass**, **Allow**, 或 **Deny**。	集群中所有 ANP 策略的规则总数，按照 **direction** 和 **action** 分组。

Name	描述	解释
ovnkube_controller_baseline_admin_network_policies_rules	• **direction**: 指定 **Ingress** 或 **Egress**。 • **action** ：指定 **Allow** 或 **Deny**。	集群中所有 BANP 策略的规则总数，按照 **direction** 和 **action** 分组。
ovnkube_controller_admin_network_policies_db_objects	**table_name** ：指定 **ACL** 或 **Address_Set**	集群中所有 ANP 创建的 OVN 北向数据库(nbdb)对象的总数，按照 **table_name** 分组。
ovnkube_controller_baseline_admin_network_policies_db_objects	**table_name** ：指定 **ACL** 或 **Address_Set**	集群中所有 BANP 创建的 OVN 北向数据库(nbdb)对象的总数，按照 **table_name** 分组。

6.2.4. AdminNetworkPolicy 的出口节点和网络对等点

本节介绍 节点 和网络 对等点。管理员可以使用本节中的示例来设计 AdminNetworkPolicy 和 BaselineAdminNetworkPolicy，以控制其集群中的北向流量。

6.2.4.1. AdminNetworkPolicy 和 BaselineAdminNetworkPolicy 的北向流量控制

除了支持 east-west 流量控制外，ANP 和 BANP 还允许管理员控制其北向流量，使集群或流量离开集群或流量到集群中的其他节点。最终用户可以执行以下操作：

- **使用 节点 出口对等点实现对集群节点的出口流量控制**
- **使用节点或网络 出口对等对 Kubernetes API 服务器实施出口流量控制**
- **使用网络对等点对集群外的外部目的地实施出口流量控制**

注意

对于 ANP 和 BANP，只能为出口规则指定节点和网络对等点。

6.2.4.1.1. 使用节点 peer 控制到集群节点的出口流量

使用 节点 对等管理员可以控制从 pod 到集群中节点的出口流量。这样做的好处是，您不必在向集群添加或删除节点时更改策略。

在以下示例中，通过使用节点选择器，允许任何带有 restricted, confidential, 或 internal 级别安全的命名空间发送的、端口 6443 上的到 Kubernetes API 服务器的出口流量。它还拒绝来自带有 restricted, confidential, or internal 安全级别的任何命名空间的、到您的集群中的所有 worker 节点的流量。

例 6.8. 使用 nodes 对等的 ANP Allow egress 示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: egress-security-allow
```

```
spec:
  egress:
  - action: Deny
    to:
    - nodes:
        matchExpressions:
        - key: node-role.kubernetes.io/worker
          operator: Exists
  - action: Allow
    name: allow-to-kubernetes-api-server-and-engr-dept-pods
    ports:
    - portNumber:
        port: 6443
        protocol: TCP
    to:
    - nodes: 1
        matchExpressions:
        - key: node-role.kubernetes.io/control-plane
          operator: Exists
    - pods: 2
        namespaceSelector:
          matchLabels:
            dept: engr
        podSelector: {}
  priority: 55
  subject: 3
    namespaces:
      matchExpressions:
      - key: security 4
        operator: In
        values:
        - restricted
        - confidential
        - internal
```

1 使用 matchExpressions 字段指定集群中的节点或一组节点。

2 指定使用 dept: engr 标记的所有 pod。

3 指定 ANP 的主题，其中包含任何与网络策略使用的标签匹配的命名空间。示例匹配任何带有 security 的级别为 restricted, confidential, 或 internal 级别的命名空间。

4 为 matchExpressions 字段指定键/值对。

6.2.4.1.2. 使用网络对等控制到外部目的地的出口流量

集群管理员可以使用网络对等中的 CIDR 范围，并应用一个策略来控制离开 pod 的出口流量，并进入通过 network 字段指定的 CIDR 范围内配置的 IP 地址的目标。

以下示例使用网络对等，并组合了 ANP 和 BANP 策略来限制出口流量。

重要

请谨慎使用 namespace 字段中的空选择器({})。使用空选择器时，它还选择 OpenShift 命名空间。

如果您在 ANP 或 BANP Deny 规则中使用 0.0.0.0/0 的值，您必须在将 Deny 设置为 0.0.0.0/0 前将更高的优先级 ANP Allow 规则设置为所需的目的地。

例 6.9. 使用网络对等点的 ANP 和 BANP 示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: network-as-egress-peer
spec:
  priority: 70
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift
namespaces as well.
  egress:
  - name: "deny-egress-to-external-dns-servers"
    action: "Deny"
    to:
    - networks: (1)
      - 8.8.8.8/32
      - 8.8.4.4/32
      - 208.67.222.222/32
    ports:
      - portNumber:
          protocol: UDP
          port: 53
  - name: "allow-all-egress-to-intranet"
    action: "Allow"
    to:
    - networks: (2)
      - 89.246.180.0/22
      - 60.45.72.0/22
  - name: "allow-all-intra-cluster-traffic"
    action: "Allow"
    to:
    - namespaces: {} # Use the empty selector with caution because it also selects
OpenShift namespaces as well.
  - name: "pass-all-egress-to-internet"
    action: "Pass"
    to:
    - networks:
      - 0.0.0.0/0 (3)
---
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift
```

```
    namespaces as well.
  egress:
  - name: "deny-all-egress-to-internet"
    action: "Deny"
    to:
    - networks:
      - 0.0.0.0/0 4
---
```

1 使用网络指定集群外的 CIDR 网络范围。

2 指定来自资源的集群内流量的 CIDR 范围。

3 4 通过将 networks 值设置为 0.0.0.0/0 来指定到任何项的 Deny egress。在将 Deny 设置为 0.0.0.0/0 之前，请确保具有较高的优先级 Allow 规则到所需的目的地，因为这将拒绝包括到 Kubernetes API 和 DNS 服务器的所有流量。

使用网络对等来整合 network-as-egress-peer ANP 和默认的 BANP 来强制以下 egress 策略：

- 所有 pod 都无法通过列出的 IP 地址与外部 DNS 服务器进行通信。
- 所有 pod 都可以与公司的其他内部网通信。
- 所有 pod 都可以与其他 pod、节点和服务通信。
- 所有 pod 都无法与互联网通信。将最后一个 ANP Pass 规则与强大的 BANP Deny 规则合并会创建一个保护策略来保护集群中的流量。

6.2.4.1.3. 一起使用节点对等和网络对等

集群管理员可以将节点和网络对等组合到 ANP 和 BANP 策略中。

例 6.10. 节点和网络对等示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: egress-peer-1 1
spec:
  egress: 2
  - action: "Allow"
    name: "allow-egress"
    to:
    - nodes:
        matchExpressions:
        - key: worker-group
          operator: In
          values:
          - workloads # Egress traffic from nodes with label worker-group: workloads is
allowed.
    - networks:
      - 104.154.164.170/32
    - pods:
```

```
      namespaceSelector:
        matchLabels:
          apps: external-apps
      podSelector:
        matchLabels:
          app: web # This rule in the policy allows the traffic directed to pods labeled apps:
web in projects with apps: external-apps to leave the cluster.
  - action: "Deny"
    name: "deny-egress"
    to:
    - nodes:
        matchExpressions:
        - key: worker-group
          operator: In
          values:
          - infra # Egress traffic from nodes with label worker-group: infra is denied.
    - networks:
      - 104.154.164.160/32 # Egress traffic to this IP address from cluster is denied.
    - pods:
        namespaceSelector:
          matchLabels:
            apps: internal-apps
        podSelector: {}
  - action: "Pass"
    name: "pass-egress"
    to:
    - nodes:
        matchExpressions:
        - key: node-role.kubernetes.io/worker
          operator: Exists # All other egress traffic is passed to NetworkPolicy or BANP for
evaluation.
  priority: 30 (3)
  subject: (4)
    namespaces:
      matchLabels:
        apps: all-apps
```

1. 指定策略的名称。
2. 对于节点和网络对等，您只能在 ANP 中使用北向流量控制作为egress。
3. 指定 ANP 的优先级，确定应评估它们的顺序。低的优先级规则具有更高的优先权。ANP 接受 0-99 的值，0 为最高优先级，99 为最低优先级。
4. 指定集群中要应用策略的 pod 集合。在示例中，所有命名空间中带有 apps: all-apps 标签的任何 pod 都是策略的主题。

6.2.5. AdminNetworkPolicy 故障排除

6.2.5.1. 检查 ANP 的创建

要检查您的 AdminNetworkPolicy (ANP) 和 BaselineAdminNetworkPolicy (BANP) 是否已正确创建，请检查以下命令的状态输出： oc describe ap 或 oc describe banp。

正常状态表示 OVN DB plumbing was successful 和 SetupSucceeded。

例 6.11. 具有良好状态的 ANP 示例

```
...
Conditions:
Last Transition Time:  2024-06-08T20:29:00Z
Message:               Setting up OVN DB plumbing was successful
Reason:                SetupSucceeded
Status:                True
Type:                  Ready-In-Zone-ovn-control-plane Last Transition Time:  2024-06-
08T20:29:00Z
Message:               Setting up OVN DB plumbing was successful
Reason:                SetupSucceeded
Status:                True
Type:                  Ready-In-Zone-ovn-worker
Last Transition Time:  2024-06-08T20:29:00Z
Message:               Setting up OVN DB plumbing was successful
Reason:                SetupSucceeded
Status:                True
Type:                  Ready-In-Zone-ovn-worker2
...
```

如果 Plumbing 失败，则会从相应的区控制器报告错误。

例 6.12. 带有错误状态和错误消息的 ANP 示例

```
...
Status:
  Conditions:
    Last Transition Time:  2024-06-25T12:47:44Z
    Message:               error attempting to add ANP cluster-control with priority 600 because,
OVNK only supports priority ranges 0-99
    Reason:                SetupFailed
    Status:                False
    Type:                  Ready-In-Zone-example-worker-1.example.example-org.net
    Last Transition Time:  2024-06-25T12:47:45Z
    Message:               error attempting to add ANP cluster-control with priority 600 because,
OVNK only supports priority ranges 0-99
    Reason:                SetupFailed
    Status:                False
    Type:                  Ready-In-Zone-example-worker-0.example.example-org.net
    Last Transition Time:  2024-06-25T12:47:44Z
    Message:               error attempting to add ANP cluster-control with priority 600 because,
OVNK only supports priority ranges 0-99
    Reason:                SetupFailed
    Status:                False
    Type:                  Ready-In-Zone-example-ctlplane-1.example.example-org.net
    Last Transition Time:  2024-06-25T12:47:44Z
    Message:               error attempting to add ANP cluster-control with priority 600 because,
OVNK only supports priority ranges 0-99
    Reason:                SetupFailed
    Status:                False
    Type:                  Ready-In-Zone-example-ctlplane-2.example.example-org.net
```

```
    Last Transition Time:  2024-06-25T12:47:44Z
    Message:               error attempting to add ANP cluster-control with priority 600 because,
OVNK only supports priority ranges 0-99
    Reason:                SetupFailed
    Status:                False
    Type:                  Ready-In-Zone-example-ctlplane-0.example.example-org.net
    ```
```

有关 nbctl 命令，请参见以下部分来帮助排除不成功的策略。

#### 6.2.5.1.1. 为 ANP 和 BANP 使用 nbctl 命令

要对不成功的设置进行故障排除，请首先查看 OVN 北向数据库 (nbdb) 对象，包括 ACL、AdressSet 和 Port_Group。要查看 nbdb，您需要在该节点上的 pod 内部查看该节点数据库中的对象。

**先决条件**

- 使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。

**注意**

要在集群中运行 ovn nbctl 命令，您必须在相关节点的 "nbdb" 中打开远程 shell。

以下策略用于生成输出。

例 6.13. 用于生成输出的 AdminNetworkPolicy

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: cluster-control
spec:
  priority: 34
  subject:
    namespaces:
      matchLabels:
        anp: cluster-control-anp # Only namespaces with this label have this ANP
  ingress:
  - name: "allow-from-ingress-router" # rule0
    action: "Allow"
    from:
    - namespaces:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
  - name: "allow-from-monitoring" # rule1
    action: "Allow"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: openshift-monitoring
    ports:
```
```

```
    - portNumber:
        protocol: TCP
        port: 7564
    - namedPort: "scrape"
  - name: "allow-from-open-tenants" # rule2
    action: "Allow"
    from:
    - namespaces: # open tenants
        matchLabels:
          tenant: open
  - name: "pass-from-restricted-tenants" # rule3
    action: "Pass"
    from:
    - namespaces: # restricted tenants
        matchLabels:
          tenant: restricted
  - name: "default-deny" # rule4
    action: "Deny"
    from:
    - namespaces: {} # Use the empty selector with caution because it also selects
OpenShift namespaces as well.
  egress:
  - name: "allow-to-dns" # rule0
    action: "Allow"
    to:
    - pods:
        namespaceSelector:
          matchlabels:
            kubernetes.io/metadata.name: openshift-dns
        podSelector:
          matchlabels:
            app: dns
    ports:
    - portNumber:
        protocol: UDP
        port: 5353
  - name: "allow-to-kapi-server" # rule1
    action: "Allow"
    to:
    - nodes:
        matchExpressions:
        - key: node-role.kubernetes.io/control-plane
          operator: Exists
    ports:
    - portNumber:
        protocol: TCP
        port: 6443
  - name: "allow-to-splunk" # rule2
    action: "Allow"
    to:
    - namespaces:
        matchlabels:
          tenant: splunk
    ports:
    - portNumber:
        protocol: TCP
```

```
      port: 8991
    - portNumber:
      protocol: TCP
      port: 8992
  - name: "allow-to-open-tenants-and-intranet-and-worker-nodes" # rule3
    action: "Allow"
    to:
    - nodes: # worker-nodes
      matchExpressions:
      - key: node-role.kubernetes.io/worker
        operator: Exists
    - networks: # intranet
      - 172.29.0.0/30
      - 10.0.54.0/19
      - 10.0.56.38/32
      - 10.0.69.0/24
    - namespaces: # open tenants
      matchLabels:
        tenant: open
  - name: "pass-to-restricted-tenants" # rule4
    action: "Pass"
    to:
    - namespaces: # restricted tenants
      matchLabels:
        tenant: restricted
  - name: "default-deny"
    action: "Deny"
    to:
    - networks:
      - 0.0.0.0/0
```

流程

1. 运行以下命令，使用节点信息列出 pod：

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

输出示例

```
NAME                                  READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-5c95487779-8k9fd   2/2     Running   0          34m   10.0.0.5     ci-
ln-0tv5gg2-72292-6sjw5-master-0   <none>           <none>
ovnkube-control-plane-5c95487779-v2xn8   2/2     Running   0          34m   10.0.0.3     ci-
ln-0tv5gg2-72292-6sjw5-master-1   <none>           <none>
ovnkube-node-524dt                    8/8     Running   0          33m   10.0.0.4     ci-ln-
0tv5gg2-72292-6sjw5-master-2   <none>           <none>
ovnkube-node-gbwr9                    8/8     Running   0          24m   10.0.128.4   ci-ln-
0tv5gg2-72292-6sjw5-worker-c-s9gqt   <none>           <none>
ovnkube-node-h4fpx                    8/8     Running   0          33m   10.0.0.5     ci-ln-
0tv5gg2-72292-6sjw5-master-0   <none>           <none>
ovnkube-node-j4hzw                    8/8     Running   0          24m   10.0.128.2   ci-ln-
0tv5gg2-72292-6sjw5-worker-a-hzbh5   <none>           <none>
ovnkube-node-wdhgv                    8/8     Running   0          33m   10.0.0.3     ci-ln-
```

```
0tv5gg2-72292-6sjw5-master-1        <none>          <none>
ovnkube-node-wfncn                  8/8     Running   0          24m   10.0.128.3   ci-ln-
0tv5gg2-72292-6sjw5-worker-b-5bb7f   <none>          <none>
```

2. 运行以下命令，进入 pod 以查看北向数据库：

```
$ oc rsh -c nbdb -n openshift-ovn-kubernetes ovnkube-node-524dt
```

3. 运行以下命令来查看 ACL nbdb ：

```
$ ovn-nbctl find ACL 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

其中, cluster-control

指定您要故障排除的 AdminNetworkPolicy 的名称。

AdminNetworkPolicy

指定类型: AdminNetworkPolicy 或 BaselineAdminNetworkPolicy。

例 6.14. ACL 的输出示例

```
_uuid               : 0d5e4722-b608-4bb1-b625-23c323cc9926
action              : allow-related
direction           : to-lport
external_ids        : {direction=Ingress, gress-index="2", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:2:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label               : 0
log                 : false
match               : "outport == @a14645450421485494999 && ((ip4.src ==
$a13730899355151937870))"
meter               : acl-logging
name                : "ANP:cluster-control:Ingress:2"
options             : {}
priority            : 26598
severity            : []
tier                : 1

_uuid               : b7be6472-df67-439c-8c9c-f55929f0a6e0
action              : drop
direction           : from-lport
external_ids        : {direction=Egress, gress-index="5", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:5:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label               : 0
log                 : false
match               : "inport == @a14645450421485494999 && ((ip4.dst ==
$a11452480169090787059))"
meter               : acl-logging
name                : "ANP:cluster-control:Egress:5"
options             : {apply-after-lb="true"}
```

```
priority            : 26595
severity            : []
tier                : 1

_uuid               : 5a6e5bb4-36eb-4209-b8bc-c611983d4624
action              : pass
direction           : to-lport
external_ids        : {direction=Ingress, gress-index="3", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:3:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label               : 0
log                 : false
match               : "outport == @a14645450421485494999 && ((ip4.src ==
$a764182844364804195))"
meter               : acl-logging
name                : "ANP:cluster-control:Ingress:3"
options             : {}
priority            : 26597
severity            : []
tier                : 1

_uuid               : 04f20275-c410-405c-a923-0e677f767889
action              : pass
direction           : from-lport
external_ids        : {direction=Egress, gress-index="4", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:4:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label               : 0
log                 : false
match               : "inport == @a14645450421485494999 && ((ip4.dst ==
$a5972452606168369118))"
meter               : acl-logging
name                : "ANP:cluster-control:Egress:4"
options             : {apply-after-lb="true"}
priority            : 26596
severity            : []
tier                : 1

_uuid               : 4b5d836a-e0a3-4088-825e-f9f0ca58e538
action              : drop
direction           : to-lport
external_ids        : {direction=Ingress, gress-index="4", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:4:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label               : 0
log                 : false
match               : "outport == @a14645450421485494999 && ((ip4.src ==
$a13814616246365836720))"
meter               : acl-logging
name                : "ANP:cluster-control:Ingress:4"
```

```
options           : {}
priority          : 26596
severity          : []
tier              : 1

_uuid             : 5d09957d-d2cc-4f5a-9ddd-b97d9d772023
action            : allow-related
direction         : from-lport
external_ids      : {direction=Egress, gress-index="2", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:2:tcp",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=tcp}
label             : 0
log               : false
match             : "inport == @a14645450421485494999 && ((ip4.dst ==
$a18396736153283155648)) && tcp && tcp.dst=={8991,8992}"
meter             : acl-logging
name              : "ANP:cluster-control:Egress:2"
options           : {apply-after-lb="true"}
priority          : 26598
severity          : []
tier              : 1

_uuid             : 1a68a5ed-e7f9-47d0-b55c-89184d97e81a
action            : allow-related
direction         : from-lport
external_ids      : {direction=Egress, gress-index="1", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:1:tcp",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=tcp}
label             : 0
log               : false
match             : "inport == @a14645450421485494999 && ((ip4.dst ==
$a10706246167277696183)) && tcp && tcp.dst==6443"
meter             : acl-logging
name              : "ANP:cluster-control:Egress:1"
options           : {apply-after-lb="true"}
priority          : 26599
severity          : []
tier              : 1

_uuid             : aa1a224d-7960-4952-bdfb-35246bafbac8
action            : allow-related
direction         : to-lport
external_ids      : {direction=Ingress, gress-index="1", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:1:tcp",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=tcp}
label             : 0
log               : false
match             : "outport == @a14645450421485494999 && ((ip4.src ==
$a6786643370959569281)) && tcp && tcp.dst==7564"
meter             : acl-logging
```

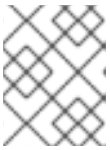
```
name              : "ANP:cluster-control:Ingress:1"
options           : {}
priority          : 26599
severity          : []
tier              : 1

_uuid             : 1a27d30e-3f96-4915-8ddd-ade7f22c117b
action            : allow-related
direction         : from-lport
external_ids      : {direction=Egress, gress-index="3", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:3:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label             : 0
log               : false
match             : "inport == @a14645450421485494999 && ((ip4.dst ==
$a10622494091691694581))"
meter             : acl-logging
name              : "ANP:cluster-control:Egress:3"
options           : {apply-after-lb="true"}
priority          : 26597
severity          : []
tier              : 1

_uuid             : b23a087f-08f8-4225-8c27-4a9a9ee0c407
action            : allow-related
direction         : from-lport
external_ids      : {direction=Egress, gress-index="0", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Egress:0:udp",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=udp}
label             : 0
log               : false
match             : "inport == @a14645450421485494999 && ((ip4.dst ==
$a13517855690389298082)) && udp && udp.dst==5353"
meter             : acl-logging
name              : "ANP:cluster-control:Egress:0"
options           : {apply-after-lb="true"}
priority          : 26600
severity          : []
tier              : 1

_uuid             : d14ed5cf-2e06-496e-8cae-6b76d5dd5ccd
action            : allow-related
direction         : to-lport
external_ids      : {direction=Ingress, gress-index="0", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:0:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-
network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-
protocol=None}
label             : 0
log               : false
match             : "outport == @a14645450421485494999 && ((ip4.src ==
$a14545668191619617708))"
```

```
meter           : acl-logging
name            : "ANP:cluster-control:Ingress:0"
options         : {}
priority        : 26600
severity        : []
tier            : 1
```



注意

ingress 和 egress 的输出显示策略在 ACL 中的逻辑。例如，每次数据包与提供的 match 匹配时会执行的 action。

a. 运行以下命令，为规则检查特定 ACL：

```
$ ovn-nbctl find ACL 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,direction=Ingress,"k8s.ovn.org/name"=cluster-
control,gress-index="1"}'
```

其中, cluster-control

指定 ANP 的名称。

入口

指定流量的 direction 为类型 Ingress 或 Egress。

1

指定要查看的规则。

对于示名为 cluster-control 的 ANP 示例，其priority 是 34，以下是 Ingress rule 1 的示例输出：

例 6.15. 输出示例

```
_uuid           : aa1a224d-7960-4952-bdfb-35246bafbac8
action          : allow-related
direction       : to-lport
external_ids    : {direction=Ingress, gress-index="1",
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:1:tcp", "k8s.ovn.org/name"=cluster-control,
"k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy, port-policy-protocol=tcp}
label           : 0
log             : false
match           : "outport == @a14645450421485494999 && ((ip4.src ==
$a6786643370959569281)) && tcp && tcp.dst==7564"
meter           : acl-logging
name            : "ANP:cluster-control:Ingress:1"
options         : {}
priority        : 26599
severity        : []
tier            : 1
```

4. 运行以下命令查看 nbdb 中的地址集：

```
$ ovn-nbctl find Address_Set 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

例 6.16. Address_Set 的输出示例

```
_uuid               : 56e89601-5552-4238-9fc3-8833f5494869
addresses           : ["192.168.194.135", "192.168.194.152", "192.168.194.193",
"192.168.194.254"]
external_ids        : {direction=Egress, gress-index="1", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:1:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a10706246167277696183

_uuid               : 7df9330d-380b-4bdb-8acd-4eddeda2419c
addresses           : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13",
"10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7",
"10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10",
"10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4",
"10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64",
"10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10",
"10.133.0.11", "10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15",
"10.133.0.16", "10.133.0.17", "10.133.0.18", "10.133.0.19", "10.133.0.20",
"10.133.0.21", "10.133.0.22", "10.133.0.23", "10.133.0.24", "10.133.0.25",
"10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29", "10.133.0.30",
"10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35",
"10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40",
"10.133.0.41", "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46",
"10.133.0.47", "10.133.0.48", "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8",
"10.133.0.9", "10.134.0.10", "10.134.0.11", "10.134.0.12", "10.134.0.13",
"10.134.0.14", "10.134.0.15", "10.134.0.16", "10.134.0.17", "10.134.0.18",
"10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22", "10.134.0.23",
"10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28",
"10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34",
"10.134.0.35", "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4",
"10.134.0.42", "10.134.0.9", "10.135.0.10", "10.135.0.11", "10.135.0.12",
"10.135.0.13", "10.135.0.14", "10.135.0.15", "10.135.0.16", "10.135.0.17",
"10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24", "10.135.0.26",
"10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40", "10.135.0.41",
"10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6", "10.135.0.7",
"10.135.0.8", "10.135.0.9"]
external_ids        : {direction=Ingress, gress-index="4", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:4:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a13814616246365836720

_uuid               : 84d76f13-ad95-4c00-8329-a0b1d023c289
addresses           : ["10.132.3.76", "10.135.0.44"]
external_ids        : {direction=Egress, gress-index="4", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:4:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
```

```
type"=AdminNetworkPolicy}
name                : a59724526061683691l8

_uuid               : 0c53e917-f7ee-4256-8f3a-9522c0481e52
addresses           : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13",
"10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7",
"10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10",
"10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4",
"10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64",
"10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10",
"10.133.0.11", "10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15",
"10.133.0.16", "10.133.0.17", "10.133.0.18", "10.133.0.19", "10.133.0.20",
"10.133.0.21", "10.133.0.22", "10.133.0.23", "10.133.0.24", "10.133.0.25",
"10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29", "10.133.0.30",
"10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35",
"10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40",
"10.133.0.41", "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46",
"10.133.0.47", "10.133.0.48", "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8",
"10.133.0.9", "10.134.0.10", "10.134.0.11", "10.134.0.12", "10.134.0.13",
"10.134.0.14", "10.134.0.15", "10.134.0.16", "10.134.0.17", "10.134.0.18",
"10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22", "10.134.0.23",
"10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28",
"10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34",
"10.134.0.35", "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4",
"10.134.0.42", "10.134.0.9", "10.135.0.10", "10.135.0.11", "10.135.0.12",
"10.135.0.13", "10.135.0.14", "10.135.0.15", "10.135.0.16", "10.135.0.17",
"10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24", "10.135.0.26",
"10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40", "10.135.0.41",
"10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6", "10.135.0.7",
"10.135.0.8", "10.135.0.9"]
external_ids        : {direction=Egress, gress-index="2", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:2:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a18396736153283155648

_uuid               : 5228bf1b-dfd8-40ec-bfa8-95c5bf9aded9
addresses           : []
external_ids        : {direction=Ingress, gress-index="0", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:0:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a14545668191619617708

_uuid               : 46530d69-70da-4558-8c63-884ec9dc4f25
addresses           : ["10.132.2.10", "10.132.2.5", "10.132.2.6", "10.132.2.7",
"10.132.2.8", "10.132.2.9", "10.133.0.47", "10.134.0.33", "10.135.0.10", "10.135.0.11",
"10.135.0.12", "10.135.0.19", "10.135.0.24", "10.135.0.7", "10.135.0.8", "10.135.0.9"]
external_ids        : {direction=Ingress, gress-index="1", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:1:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a67866433709595692l1
```

```
_uuid               : 65fdcdea-0b9f-4318-9884-1b51d231ad1d
addresses           : ["10.132.3.72", "10.135.0.42"]
external_ids        : {direction=Ingress, gress-index="2", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:2:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a13730899355151937870

_uuid               : 73eabdb0-36bf-4ca3-b66d-156ac710df4c
addresses           : ["10.0.32.0/19", "10.0.56.38/32", "10.0.69.0/24", "10.132.3.72",
"10.135.0.42", "172.29.0.0/30", "192.168.194.103", "192.168.194.2"]
external_ids        : {direction=Egress, gress-index="3", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:3:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a10622494091691694581

_uuid               : 50cdbef2-71b5-474b-914c-6fcd1d7712d3
addresses           : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13",
"10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7",
"10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10",
"10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4",
"10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64",
"10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10",
"10.133.0.11", "10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15",
"10.133.0.16", "10.133.0.17", "10.133.0.18", "10.133.0.19", "10.133.0.20",
"10.133.0.21", "10.133.0.22", "10.133.0.23", "10.133.0.24", "10.133.0.25",
"10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29", "10.133.0.30",
"10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35",
"10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40",
"10.133.0.41", "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46",
"10.133.0.47", "10.133.0.48", "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8",
"10.133.0.9", "10.134.0.10", "10.134.0.11", "10.134.0.12", "10.134.0.13",
"10.134.0.14", "10.134.0.15", "10.134.0.16", "10.134.0.17", "10.134.0.18",
"10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22", "10.134.0.23",
"10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28",
"10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34",
"10.134.0.35", "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4",
"10.134.0.42", "10.134.0.9", "10.135.0.10", "10.135.0.11", "10.135.0.12",
"10.135.0.13", "10.135.0.14", "10.135.0.15", "10.135.0.16", "10.135.0.17",
"10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24", "10.135.0.26",
"10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40", "10.135.0.41",
"10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6", "10.135.0.7",
"10.135.0.8", "10.135.0.9"]
external_ids        : {direction=Egress, gress-index="0", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:0:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a13517855690389298082

_uuid               : 32a42f32-2d11-43dd-979d-a56d7ee6aa57
addresses           : ["10.132.3.76", "10.135.0.44"]
```

```
external_ids        : {direction=Ingress, gress-index="3", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Ingress:3:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a764182844364804195

_uuid               : 8fd3b977-6e1c-47aa-82b7-e3e3136c4a72
addresses           : ["0.0.0.0/0"]
external_ids        : {direction=Egress, gress-index="5", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:5:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a11452480169090787059
```

a. 运行以下命令，检查规则的具体地址集：

```
$ ovn-nbctl find Address_Set 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,direction=Egress,"k8s.ovn.org/name"=cluster-
control,gress-index="5"}'
```

例 6.17. Address_Set 的输出示例

```
_uuid               : 8fd3b977-6e1c-47aa-82b7-e3e3136c4a72
addresses           : ["0.0.0.0/0"]
external_ids        : {direction=Egress, gress-index="5", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:5:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy}
name                : a11452480169090787059
```

5. 运行以下命令查看 nbdb 中的端口组：

```
$ ovn-nbctl find Port_Group 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

例 6.18. Port_Group 的输出示例

```
_uuid               : f50acf71-7488-4b9a-b7b8-c8a024e99d21
acls                : [04f20275-c410-405c-a923-0e677f767889, 0d5e4722-b608-4bb1-b625-
23c323cc9926, 1a27d30e-3f96-4915-8ddd-ade7f22c117b, 1a68a5ed-e7f9-47d0-b55c-
89184d97e81a, 4b5d836a-e0a3-4088-825e-f9f0ca58e538, 5a6e5bb4-36eb-4209-b8bc-
c611983d4624, 5d09957d-d2cc-4f5a-9ddd-b97d9d772023, aa1a224d-7960-4952-bdfb-
35246bafbac8, b23a087f-08f8-4225-8c27-4a9a9ee0c407, b7be6472-df67-439c-8c9c-
f55929f0a6e0, d14ed5cf-2e06-496e-8cae-6b76d5dd5ccd]
external_ids        : {"k8s.ovn.org/id"="default-network-
controller:AdminNetworkPolicy:cluster-control", "k8s.ovn.org/name"=cluster-
control, "k8s.ovn.org/owner-controller"=default-network-controller,
"k8s.ovn.org/owner-type"=AdminNetworkPolicy}
```

```
name            : a14645450421485494999
ports           : [5e75f289-8273-4f8a-8798-8c10f7318833, de7e1b71-6184-445d-93e7-
b20acadf41ea]
```

6.2.5.2. 其他资源

- 使用 ovnkube-trace 追踪 Openflow
- OVN-Kubernetes 故障排除

6.3. 网络策略

6.3.1. 关于网络策略

作为集群管理员，您可以定义网络策略以限制到集群中的 pod 的网络通讯。

6.3.1.1. 关于网络策略

在使用支持 Kubernetes 网络策略的网络插件的集群中，网络隔离完全由 NetworkPolicy 对象控制。在 OpenShift Container Platform 4.16 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。

警告

网络策略不适用于主机网络命名空间。启用主机网络的 Pod 不受网络策略规则的影响。但是，连接到 host-networked pod 的 pod 会受到网络策略规则的影响。

网络策略无法阻止来自 localhost 或来自其驻留的节点的流量。

默认情况下，项目中的所有 pod 都可被其他 pod 和网络端点访问。要在一个项目中隔离一个或多个 Pod，您可以在该项目中创建 NetworkPolicy 对象来指示允许的入站连接。项目管理员可以在自己的项目中创建和删除 NetworkPolicy 对象。

如果一个 pod 由一个或多个 NetworkPolicy 对象中的选择器匹配，那么该 pod 将只接受至少被其中一个 NetworkPolicy 对象所允许的连接。未被任何 NetworkPolicy 对象选择的 pod 可以完全访问。

网络策略仅适用于 TCP、UDP、ICMP 和 SCTP 协议。其他协议不会受到影响。

以下示例 NetworkPolicy 对象演示了支持不同的情景：

- 拒绝所有流量：
  要使项目默认为拒绝流量，请添加一个匹配所有 pod 但不接受任何流量的 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
```

```
spec:
  podSelector: {}
  ingress: []
```

- 只允许 OpenShift Container Platform Ingress Controller 的连接：
要使项目只允许 OpenShift Container Platform Ingress Controller 的连接，请添加以下 NetworkPolicy 对象。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

- 只接受项目中 pod 的连接：
要使 pod 接受同一项目中其他 pod 的连接，但拒绝其他项目中所有 pod 的连接，请添加以下 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
```

- 仅允许基于 pod 标签的 HTTP 和 HTTPS 流量：
要对带有特定标签（以下示例中的 role=frontend）的 pod 仅启用 HTTP 和 HTTPS 访问，请添加类似如下的 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- 使用命名空间和 pod 选择器接受连接：
  要通过组合使用命名空间和 pod 选择器来匹配网络流量,您可以使用类似如下的 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
        podSelector:
          matchLabels:
            name: test-pods
```

NetworkPolicy 对象是可添加的；也就是说，您可以组合多个 NetworkPolicy 对象来满足复杂的网络要求。

例如，对于以上示例中定义的 NetworkPolicy 对象，您可以在同一个项目中定义 allow-same-namespace 和 allow-http-and-https 策略。因此，允许带有标签 role=frontend 的 pod 接受每一策略所允许的任何连接。即，任何端口上来自同一命名空间中的 pod 的连接，以及端口 80 和 443 上的来自任意命名空间中 pod 的连接。

6.3.1.1.1. 使用 allow-from-router 网络策略

使用以下 NetworkPolicy 来允许外部流量，而不考虑路由器配置：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" (1)
  podSelector: {}
  policyTypes:
  - Ingress
```

(1) policy-group.network.openshift.io/ingress:"" 标签支持 OpenShift-SDN 和 OVN-Kubernetes。

6.3.1.1.2. 使用 allow-from-hostnetwork 网络策略

添加以下 allow-from-hostnetwork NetworkPolicy 对象来指示来自主机网络 pod 的流量：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

6.3.1.2. 使用 OpenShift SDN 优化网络策略

使用一个网络策略来通过 pod 上的不同标签来在命名空间中将不同 pod 进行隔离。

将 NetworkPolicy 对象应用到单一命名空间中的大量 pod 时，效率较低。因为 Pod 标签不存在于 IP 地址一级，因此网络策略会为使用 podSelector 选择的每个 pod 之间生成单独的 Open vSwitch（OVS）流量规则 。

例如，在一个 NetworkPolicy 对象中，如果 spec podSelector 和 ingress podSelector 每个都匹配 200 个 pod，则会产生 40,000 (200*200) OVS 流规则。这可能会减慢节点的速度。

在设计您的网络策略时，请参考以下指南：

- 使用命名空间使其包含需要隔离的 pod 组，可以减少 OVS 流规则数量。
  使用 namespaceSelector 或空 podSelector 选择整个命名空间的NetworkPolicy 对象会只生成一个与命名空间的 VXLAN 虚拟网络 ID（VNID）匹配的 OVS 流量规则。

- 保留不需要在原始命名空间中隔离的 pod，并将需要隔离的 pod 移到一个或多个不同的命名空间中。

- 创建额外的目标跨命名空间网络策略，以允许来自不同隔离的 pod 的特定流量。

6.3.1.3. 使用 OVN-Kubernetes 网络插件优化网络策略

在设计您的网络策略时，请参考以下指南：

- 对于具有相同 spec.podSelector spec 的网络策略，使用带有多个 ingress 或 egress 规则的一个网络策略比带有 ingress 或 egress 子集的多个网络策略更高效。

- 每个基于 podSelector 或 namespaceSelector spec 的 ingress 或 egress 规则会生成一个的 OVS 流数量，它与由网络策略选择的 pod 数量 + 由 ingress 或 egress 选择的 pod 数量成比例因此，最好使用在一个规则中可以选择您所需的 pod 的 podSelector 或 namespaceSelector 规格，而不是为每个 pod 创建单独的规则。
  例如，以下策略包含两个规则：

  ```
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: test-network-policy
  spec:
    podSelector: {}
  ```

```
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

以下策略表示这两个规则与以下相同的规则：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - {key: role, operator: In, values: [frontend, backend]}
```

相同的指南信息适用于 spec.podSelector spec。如果不同的网络策略有相同的 ingress 或 egress 规则，则创建一个带有通用的 spec.podSelector spec 可能更有效率。例如，以下两个策略有不同的规则：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy2
spec:
  podSelector:
    matchLabels:
      role: client
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

以下网络策略将这两个相同的规则作为一个：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

当只有多个选择器表示为一个选择器时，您可以应用此优化。如果选择器基于不同的标签，则可能无法应用此优化。在这些情况下，请考虑为网络策略优化应用一些新标签。

6.3.1.4. 后续步骤

- 创建网络策略
- 可选：为项目定义默认网络策略

6.3.1.5. 其他资源

- 项目和命名空间
- 使用网络策略配置多租户隔离
- 网络策略 API

6.3.2. 创建网络策略

作为具有 admin 角色的用户，您可以为命名空间创建网络策略。

6.3.2.1. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
```

```
ports: 4
- protocol: TCP
  port: 27017
```

1. NetworkPolicy 对象的名称。
2. 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。
3. 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。
4. 接受流量的一个或多个目标端口的列表。

6.3.2.2. 使用 CLI 创建网络策略

要定义细致的规则来描述集群中命名空间允许的入口或出口网络流量，您可以创建一个网络策略。

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略规则：
   a. 创建一个 <policy_name>.yaml 文件：

   ```
   $ touch <policy_name>.yaml
   ```

   其中：

   <policy_name>
   指定网络策略文件名。

   b. 在您刚才创建的文件中定义网络策略，如下例所示：

   拒绝来自所有命名空间中的所有 pod 的入口流量

   这是一个基本的策略，阻止配置其他网络策略所允许的跨 pod 流量以外的所有跨 pod 网络。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   ```

```
metadata:
  name: deny-by-default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress: []
```

允许来自所有命名空间中的所有 pod 的入口流量

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

允许从特定命名空间中到一个 pod 的入口流量

此策略允许流量从在 namespace-y 中运行的容器集到标记 pod-a 的 pod。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-traffic-pod
spec:
  podSelector:
   matchLabels:
     pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

2. 运行以下命令来创建网络策略对象：

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

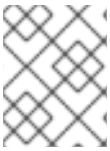
其中：

<policy_name>
指定网络策略文件名。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

```
networkpolicy.networking.k8s.io/deny-by-default created
```



注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式创建网络策略。

6.3.2.3. 创建默认拒绝所有网络策略

这是一个基本的策略，阻止其他部署网络策略允许的网络流量以外的所有跨 pod 网络。此流程强制使用默认 deny-by-default 策略。

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建以下 YAML，以定义 deny-by-default 策略，以拒绝所有命名空间中的所有 pod 的入口流量。将 YAML 保存到 deny-by-default.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: default (1)
spec:
  podSelector: {} (2)
  ingress: [] (3)
```

(1) namespace：default 将此策略部署到 default 命名空间。

(2) podSelector: 为空，这意味着它与所有 pod 匹配。因此，该策略适用于 default 命名空间中的所有 pod。

(3) 没有指定 ingress 规则。这会导致传入的流量丢弃至所有 pod。

2. 输入以下命令应用策略：

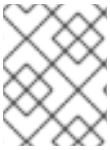
```
$ oc apply -f deny-by-default.yaml
```

输出示例

```
networkpolicy.networking.k8s.io/deny-by-default created
```

6.3.2.4. 创建网络策略以允许来自外部客户端的流量

使用 deny-by-default 策略，您可以继续配置策略，允许从外部客户端到带有标签 app=web 的 pod 的流量。



注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置策略，以直接从公共互联网允许外部服务，或使用 Load Balancer 访问 pod。只有具有标签 app=web 的 pod 才允许流量。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略，以直接从公共互联网的流量或使用负载均衡器访问 pod。将 YAML 保存到 web-allow-external.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
  namespace: default
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

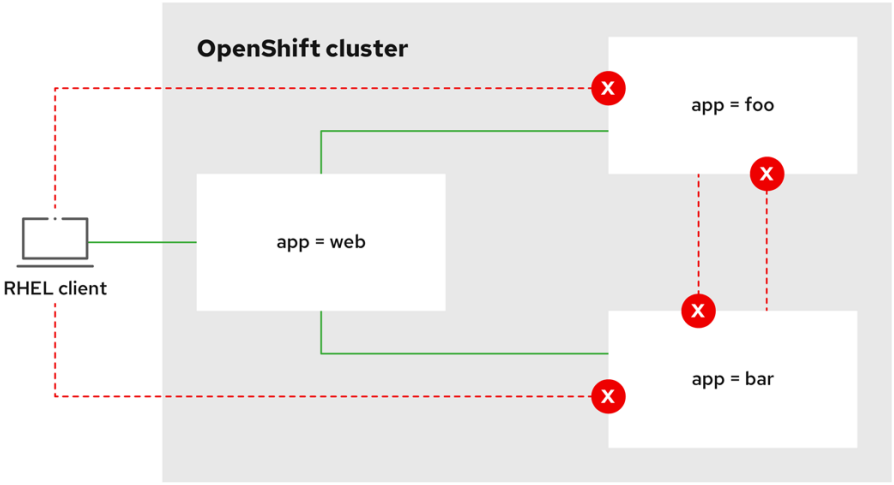
2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-external.yaml
```

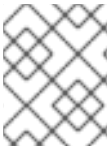
输出示例

```
networkpolicy.networking.k8s.io/web-allow-external created
```

此策略允许来自所有资源的流量，包括下图所示的外部流量：



6.3.2.5. 创建网络策略，允许从所有命名空间中到应用程序的流量



注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置允许从所有命名空间中的所有 pod 流量到特定应用程序的策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

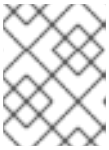
1. 创建一个策略，允许从所有命名空间中的所有 pod 流量到特定应用。将 YAML 保存到 web-allow-all-namespaces.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-namespaces
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
```

```
policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector: {} 2
```

1 仅将策略应用到 default 命名空间中的 app:web pod。

2 选择所有命名空间中的所有 pod。



注意

默认情况下，如果您省略了指定 namespaceSelector 而不是选择任何命名空间，这意味着策略只允许从网络策略部署到的命名空间的流量。

2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-all-namespaces.yaml
```

输出示例

```
networkpolicy.networking.k8s.io/web-allow-all-namespaces created
```

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --
port=80
```

2. 运行以下命令在 secondary 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. 在 shell 中运行以下命令，并观察是否允许请求：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

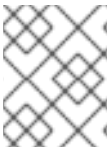
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
```

```
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

6.3.2.6. 创建网络策略，允许从一个命名空间中到应用程序的流量



注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置允许从特定命名空间中到带有 app=web 标签的 pod 的策略。您可能需要进行以下操作：

- 将流量限制为部署生产工作负载的命名空间。
- 启用部署到特定命名空间的监控工具，以从当前命名空间中提取指标。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建一个策略，允许来自特定命名空间中所有 pod 的流量，其标签为 purpose=production。将 YAML 保存到 web-allow-prod.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
```

```
  - Ingress
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        purpose: production 2
```

1 仅将策略应用到 default 命名空间中的 app:web pod。

2 将流量仅限制为具有标签 purpose=production 的命名空间中的 pod。

2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-prod.yaml
```

输出示例

```
networkpolicy.networking.k8s.io/web-allow-prod created
```

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --
port=80
```

2. 运行以下命令来创建 prod 命名空间：

```
$ oc create namespace prod
```

3. 运行以下命令来标记 prod 命名空间：

```
$ oc label namespace/prod purpose=production
```

4. 运行以下命令来创建 dev 命名空间：

```
$ oc create namespace dev
```

5. 运行以下命令来标记 dev 命名空间：

```
$ oc label namespace/dev purpose=testing
```

6. 运行以下命令在 dev 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```

7. 在 shell 中运行以下命令，并观察请求是否被阻止：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

```
wget: download timed out
```

8. 运行以下命令，在 prod 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. 在 shell 中运行以下命令，并观察是否允许请求：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

6.3.2.7. 其他资源

- 访问Web控制台
- 出口防火墙和网络策略规则的日志记录

6.3.3. 查看网络策略

以具有 admin 角色的用户，您可以查看命名空间的网络策略。

6.3.3.1. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1 NetworkPolicy 对象的名称。

2 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。

3 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。

4 接受流量的一个或多个目标端口的列表。

6.3.3.2. 使用 CLI 查看网络策略

您可以检查命名空间中的网络策略。

注意

如果使用具有 cluster-admin 角色的用户登录，您可以查看集群中的任何网络策略。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

- 列出命名空间中的网络策略：
  - 要查看命名空间中定义的网络策略对象，请输入以下命令：

    ```
    $ oc get networkpolicy
    ```

  - 可选：要检查特定的网络策略，请输入以下命令：

    ```
    $ oc describe networkpolicy <policy_name> -n <namespace>
    ```

其中：

<policy_name>
指定要检查的网络策略的名称。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

例如：

```
$ oc describe networkpolicy allow-same-namespace
```

oc describe 命令的输出

```
Name:         allow-same-namespace
Namespace:    ns1
Created on:   2021-05-24 22:28:56 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this
namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: <none>
  Not affecting egress traffic
  Policy Types: Ingress
```

注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式查看网络策略。

6.3.4. 编辑网络策略

作为具有 admin 角色的用户，您可以编辑命名空间的现有网络策略。

6.3.4.1. 编辑网络策略

您可以编辑命名空间中的网络策略。

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中编辑网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。

- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

1. 可选：要列出一个命名空间中的网络策略对象，请输入以下命令：

```
$ oc get networkpolicy
```

其中：

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

2. 编辑网络策略对象。
   - 如果您在文件中保存了网络策略定义，请编辑该文件并进行必要的更改，然后输入以下命令。

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

其中：

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。
<policy_file>
指定包含网络策略的文件的名称。

   - 如果您需要直接更新网络策略对象，请输入以下命令：

```
$ oc edit networkpolicy <policy_name> -n <namespace>
```

其中：

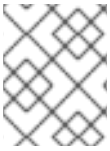
<policy_name>
指定网络策略的名称。
<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

3. 确认网络策略对象已更新。

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

其中：

<policy_name>
指定网络策略的名称。
<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。



注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或通过 Actions 菜单从 web 控制台中的策略编辑网络策略。

6.3.4.2. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy 对象的名称。
2. 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。
3. 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。
4. 接受流量的一个或多个目标端口的列表。

6.3.4.3. 其他资源

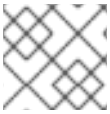
- 创建网络策略

6.3.5. 删除网络策略

以具有 admin 角色的用户，您可以从命名空间中删除网络策略。

6.3.5.1. 使用 CLI 删除网络策略

您可以删除命名空间中的网络策略。



注意

如果使用具有 cluster-admin 角色的用户登录，您可以删除集群中的任何网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

- 要删除网络策略对象，请输入以下命令：

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

其中：

<policy_name>
指定网络策略的名称。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

```
networkpolicy.networking.k8s.io/default-deny deleted
```

注意

如果使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群上以 YAML 或通过 Actions 菜单从 web 控制台中的策略删除网络策略。

6.3.6. 为项目定义默认网络策略

作为集群管理员，您可以在创建新项目时修改新项目模板，使其自动包含网络策略。如果您还没有新项目的自定义模板，则需要首先创建一个。

6.3.6.1. 为新项目修改模板

作为集群管理员，您可以修改默认项目模板，以便使用自定义要求创建新项目。

创建自己的自定义项目模板：

先决条件

- 可以使用具有 cluster-admin 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 以具有 cluster-admin 特权的用户身份登录。
2. 生成默认项目模板：

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 使用文本编辑器，通过添加对象或修改现有对象来修改生成的 template.yaml 文件。
4. 项目模板必须创建在 openshift-config 命名空间中。加载修改后的模板：

```
$ oc create -f template.yaml -n openshift-config
```

5. 使用 Web 控制台或 CLI 编辑项目配置资源。
   - 使用 Web 控制台：
     i. 导航至 Administration → Cluster Settings 页面。
     ii. 单击 Configuration 以查看所有配置资源。
     iii. 找到 Project 的条目，并点击 Edit YAML。
   - 使用 CLI：
     i. 编辑 project.config.openshift.io/cluster 资源：

```
$ oc edit project.config.openshift.io/cluster
```

6. 更新 spec 部分，使其包含 projectRequestTemplate 和 name 参数，再设置您上传的项目模板的名称。默认名称为 project-request。

   带有自定义项目模板的项目配置资源

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
# ...
spec:
  projectRequestTemplate:
    name: <template_name>
# ...
```

7. 保存更改后，创建一个新项目来验证是否成功应用了您的更改。

6.3.6.2. 在新项目模板中添加网络策略

作为集群管理员，您可以在新项目的默认模板中添加网络策略。OpenShift Container Platform 将自动创建项目中模板中指定的所有 NetworkPolicy 对象。

先决条件

- 集群使用支持 NetworkPolicy 对象的默认 CNI 网络插件，如设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您需要使用具有 cluster-admin 权限的用户登陆到集群。
- 您必须已为新项目创建了自定义的默认项目模板。

流程

1. 运行以下命令来编辑新项目的默认模板：

```
$ oc edit template <project_template> -n openshift-config
```

将 <project_template> 替换为您为集群配置的缺省模板的名称。默认模板名称为 project-request。

2. 在模板中，将每个 NetworkPolicy 对象作为一个元素添加到 objects 参数中。objects 参数可以是一个或多个对象的集合。
在以下示例中，objects 参数集合包括几个 NetworkPolicy 对象。

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
    - Ingress
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
    policyTypes:
    - Ingress
...
```

3. 可选：通过运行以下命令创建一个新项目，来确认您的网络策略对象已被成功创建：

   a. 创建一个新项目：

```
$ oc new-project <project> 1
```

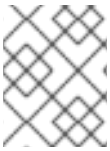
1 将 <project> 替换为您要创建的项目的名称。

b. 确认新项目模板中的网络策略对象存在于新项目中：

```
$ oc get networkpolicy
NAME                          POD-SELECTOR   AGE
allow-from-openshift-ingress   <none>        7s
allow-from-same-namespace      <none>        7s
```

6.3.7. 使用网络策略配置多租户隔离

作为集群管理员，您可以配置网络策略以为多租户网络提供隔离功能。



注意

如果使用 OpenShift SDN 网络插件，请按照本节所述配置网络策略，提供类似于多租户模式的网络隔离，但设置了网络策略模式。

6.3.7.1. 使用网络策略配置多租户隔离

您可以配置项目，使其与其他项目命名空间中的 pod 和服务分离。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。

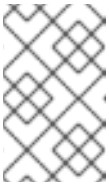
流程

1. 创建以下 NetworkPolicy 对象：

   a. 名为 allow-from-openshift-ingress 的策略。

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
  podSelector: {}
```

```
  policyTypes:
  - Ingress
EOF
```



注意

policy-group.network.openshift.io/ingress: "" 是 OpenShift SDN 的首选命名空间选择器标签。您可以使用 network.openshift.io/policy-group: ingress 命名空间选择器标签，但这是一个比较旧的用法。

b. 名为 **allow-from-openshift-monitoring** 的策略：

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c. 名为 **allow-same-namespace** 的策略：

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

d. 名为 **allow-from-kube-apiserver-operator** 的策略：

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
```

```
        kubernetes.io/metadata.name: openshift-kube-apiserver-operator
    podSelector:
      matchLabels:
        app: kube-apiserver-operator
  policyTypes:
  - Ingress
EOF
```

如需了解更多详细信息，请参阅 新的kube-apiserver-operator Webhook 控制器验证 Webhook 的健康状况。

2. 可选：要确认当前项目中存在网络策略，请输入以下命令：

```
$ oc describe networkpolicy
```

输出示例

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

6.3.7.2. 后续步骤

- 为项目定义默认网络策略

6.3.7.3. 其他资源

- OpenShift SDN 网络隔离模式

6.4. 网络安全的审计日志记录

OVN-Kubernetes 网络插件使用 Open Virtual Network (OVN) 访问控制列表 (ACL) 来管理 AdminNetworkPolicy、BaselineAdminNetworkPolicy、NetworkPolicy 和 EgressFirewall 对象。审计日志记录会公开 NetworkPolicy, EgressFirewall 和 BaselineAdminNetworkPolicy 自定义资源 (CR) 的 allow 和 deny ACL 事件。日志记录还公开 AdminNetworkPolicy (ANP) CR 的allow, deny, 和 pass ACL 的事件。

注意

审计日志记录仅适用于 OVN-Kubernetes 网络插件。

6.4.1. 审计配置

审计日志记录的配置作为 OVN-Kubernetes 集群网络配置的一部分指定。以下 YAML 演示了审计日志的默认值：

审计日志记录配置

```yaml
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
```

下表描述了审计日志的配置字段。

表 6.1. policyAuditConfig object

字段	类型	描述
rateLimit	整数	每个节点每秒生成一次的消息数量上限。默认值为每秒 20 条消息。
maxFileSize	整数	审计日志的最大大小，以字节为单位。默认值为 50000000 或 50 MB。
maxLogFiles	整数	保留的日志文件的最大数量。

字段	类型	描述
目的地	字符串	以下附加审计日志目标之一： **libc** 主机上的 journald 进程的 libc **syslog（）** 函数。 **UDP:<host>:<port>** 一个 syslog 服务器。将 **<host>:<port>** 替换为 **syslog 服务器的主机** 和端口。 **Unix:<file>** 由 **<file>** 指定的 Unix 域套接字文件。 **null** 不要将审计日志发送到任何其他目标。
syslogFacility	字符串	syslog 工具，如 as **kern**，如 RFC5424 定义。默认值为 **local0。**

6.4.2. 审计日志记录

您可以为审计日志配置目的地，如 syslog 服务器或 UNIX 域套接字。无论任何其他配置如何，审计日志始终保存到集群中的每个 OVN-Kubernetes pod 上的 /var/log/ovn/acl-audit-log。

通过使用 k8s.ovn.org/acl-logging 键注解命名空间，为每个命名空间启用审计日志记录，如下例所示：

命名空间注解示例

```
kind: Namespace
apiVersion: v1
metadata:
  name: example1
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "info",
        "allow": "info"
      }
```

日志记录格式与 RFC5424 中定义的 syslog 兼容。syslog 工具可配置，默认为 local0。日志条目示例可能类似如下：

网络策略的 ACL 拒绝日志条目示例

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_fla
```

```
gs=syn
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
```

下表描述了命名空间注解值：

表 6.2. 审计日志记录命名空间注解

注解	value
k8s.ovn.org/acl-logging	您必须至少指定 allow, deny 之一，或两者都指定来为一个命名空间启用审计日志。 **deny** 可选：指定 alert、warning、notice、info 或 debug。 **allow** 可选：指定 alert、warning、notice、info 或 debug。

6.4.3. AdminNetworkPolicy 审计日志记录

每个 AdminNetworkPolicy CR 启用了审计日志记录，使用 k8s.ovn.org/acl-logging 键注解 ANP 策略，如下例所示：

例 6.19. AdminNetworkPolicy CR 的注解示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  annotations:
    k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert", "pass" : "warning" }'
  name: anp-tenant-log
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        tenant: backend-storage # Selects all pods owned by storage tenant.
  ingress:
    - name: "allow-all-ingress-product-development-and-customer" # Product development and customer tenant ingress to backend storage.
      action: "Allow"
      from:
      - pods:
          namespaceSelector:
            matchExpressions:
            - key: tenant
              operator: In
              values:
              - product-development
              - customer
```

```
        podSelector: {}
    - name: "pass-all-ingress-product-security"
      action: "Pass"
      from:
      - namespaces:
          matchLabels:
            tenant: product-security
    - name: "deny-all-ingress" # Ingress to backend from all other pods in the cluster.
      action: "Deny"
      from:
      - namespaces: {}
  egress:
    - name: "allow-all-egress-product-development"
      action: "Allow"
      to:
      - pods:
          namespaceSelector:
            matchLabels:
              tenant: product-development
          podSelector: {}
    - name: "pass-egress-product-security"
      action: "Pass"
      to:
      - namespaces:
          matchLabels:
            tenant: product-security
    - name: "deny-all-egress" # Egress from backend denied to all other pods.
      action: "Deny"
      to:
      - namespaces: {}
```

每当特定的 OVN ACL 达到并满足日志记录注解中设置的操作条件时，都会生成日志。例如，一个事件，其中任何带有标签 **tenant: product-development** 的命名空间都访问带有标签 **tenant: backend-storage** 的命名空间，则会生成日志。

注意

ACL 日志记录限制为 60 个字符。如果您的 ANP **name** 字段较长，日志的其余部分将被截断。

以下是以下示例日志条目的方向索引：

方向	规则

方向	规则
入口	Rule0 允许从租户 **product-development** 和 **customer** 到租户 **backend-storage**; Ingress0: **Allow** Rule1 从 **product-security` 到租户 `backend-storage**; Ingress1: **Pass** 传递 Rule2 拒绝来自所有 pod 的入站流量; Ingress2: **Deny**
Egress	Rule0 允许到 **product-development**; Egress0: **Allow** Rule1 传递给 **product-security**; Egress1**Pass** Rule2 拒绝到所有其他 pod 的出口流量; Egress2: **Deny**

例 6.20. ACL 日志项示例，用于带有 Ingress:0 和 Egress:0 的名为 anp-tenant-log 的 AdminNetworkPolicy 的 Allow 操作

```
2024-06-10T16:27:45.194Z|00052|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:0", verdict=allow, severity=alert, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1a,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.26,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=57814,tp_dst=8080,tcp_flags=syn
2024-06-10T16:28:23.130Z|00059|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:0", verdict=allow, severity=alert, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:18,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.24,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=38620,tp_dst=8080,tcp_flags=ack
2024-06-10T16:28:38.293Z|00069|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Egress:0", verdict=allow, severity=alert, direction=from-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:1a,nw_src=10.128.2.25,nw_dst=10.128.2.26,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=47566,tp_dst=8080,tcp_flags=fin|ack=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=55704,tp_dst=8080,tcp_flags=ack
```

例 6.21. ACL 日志项示例，用于带有 Ingress:1 和 Egress:1 的名为 anp-tenant-log 的 AdminNetworkPolicy 的 Pass 操作

```
2024-06-10T16:33:12.019Z|00075|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:1", verdict=pass, severity=warning, direction=to-lport: tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1b,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.27,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=37394,tp_dst=8080,tcp_flags=ack
2024-06-10T16:35:04.209Z|00081|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Egress:1", verdict=pass, severity=warning, direction=from-lport:
```

```
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:1b,nw_src=10.128.2.25
,nw_dst=10.128.2.27,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=34018,tp_dst=8080,
tcp_flags=ack
```

例 6.22. ACL 日志项示例，用于带有 Ingress:2 和 Egress:2 的名为 anp-tenant-log 的 AdminNetworkPolicy 的 Deny 操作

```
2024-06-10T16:43:05.287Z|00087|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-
log:Egress:2", verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:18,nw_src=10.128.2.25,
nw_dst=10.128.2.24,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=51598,tp_dst=8080,t
cp_flags=syn
2024-06-10T16:44:43.591Z|00090|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-
log:Ingress:2", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1c,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.28,
nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=33774,tp_dst=8080,t
cp_flags=syn
```

下表描述了 ANP 注解：

表 6.3. 审计日志记录 AdminNetworkPolicy 注解

注解	value
k8s.ovn.org/acl-logging	您必须至少指定 **Allow**、**Deny** 或 **Pass** 之一才能为命名空间启用审计日志记录。 **Deny** 可选：指定 **alert**、**warning**、**notice**、**info** 或 **debug**。 **Allow** 可选：指定 **alert**、**warning**、**notice**、**info** 或 **debug**。 **Pass** 可选：指定 **alert**、**warning**、**notice**、**info** 或 **debug**。

6.4.4. BaselineAdminNetworkPolicy 审计日志记录

使用 k8s.ovn.org/acl-logging 键注解 BANP 策略，在 BaselineAdminNetworkPolicy CR 中启用审计日志记录，如下例所示：

例 6.23. BaselineAdminNetworkPolicy CR 的注解示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  annotations:
    k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert"}'
  name: default
spec:
  subject:
```

```
    namespaces:
      matchLabels:
        tenant: workloads # Selects all workload pods in the cluster.
  ingress:
  - name: "default-allow-dns" # This rule allows ingress from dns tenant to all workloads.
    action: "Allow"
    from:
    - namespaces:
        matchLabels:
          tenant: dns
  - name: "default-deny-dns" # This rule denies all ingress from all pods to workloads.
    action: "Deny"
    from:
    - namespaces: {} # Use the empty selector with caution because it also selects
OpenShift namespaces as well.
  egress:
  - name: "default-deny-dns" # This rule denies all egress from workloads. It will be applied
when no ANP or network policy matches.
    action: "Deny"
    to:
    - namespaces: {} # Use the empty selector with caution because it also selects
OpenShift namespaces as well.
```

在示例中，一个事件，其中任何带有标签 tenant: dns 的命名空间都访问带有标签 tenant: workload 的命名空间，则会生成日志。

以下是以下示例日志条目的方向索引：

方向	规则
入口	Rule0 允许从租户 **dns** 到租户 **workloads**; Ingress0: **Allow** Rule1 拒绝从所有 pod 到租户 **workloads**; Ingress1: **Deny**
Egress	Rule0 拒绝所有 pod; Egress0: **Deny**

例 6.24. ACL allow 日志项示例，用于带有 Ingress:0 的 default BANP 的 Allow 操作

```
2024-06-
10T18:11:58.263Z|00022|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=syn
2024-06-
10T18:11:58.264Z|00023|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
```

```
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=psh|ack
2024-06-
10T18:11:58.264Z|00024|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=ack
2024-06-
10T18:11:58.264Z|00025|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=ack
2024-06-
10T18:11:58.264Z|00026|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=fin|ack
2024-06-
10T18:11:58.264Z|00027|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,t
cp_flags=ack
```

例 6.25. ACL allow 日志项示例，用于带有 Egress:0 和 Ingress:1 的 default BANP 的 Allow 操作

```
2024-06-
10T18:09:57.774Z|00016|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,
nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,t
cp_flags=syn
2024-06-
10T18:09:58.809Z|00017|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,
nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,t
cp_flags=syn
2024-06-
10T18:10:00.857Z|00018|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,
nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,t
cp_flags=syn
2024-06-
10T18:10:25.414Z|00019|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,t
cp_flags=syn
```

```
2024-06-
10T18:10:26.457Z|00020|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,t
cp_flags=syn
2024-06-
10T18:10:28.505Z|00021|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,
nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,t
cp_flags=syn
```

下表描述了 BANP 注解：

表 6.4. 审计日志记录 BaselineAdminNetworkPolicy 注解

注解	value
k8s.ovn.org/acl-logging	您必须至少指定 **Allow** 或 **Deny** 之一才能为命名空间启用审计日志记录。 **Deny** 可选：指定 **alert**、**warning**、**notice**、**info** 或 **debug**。 **Allow** 可选：指定 **alert**、**warning**、**notice**、**info** 或 **debug**。

6.4.5. 为集群配置出口防火墙和网络策略审计

作为集群管理员，您可以自定义集群的审计日志。

先决条件

- 安装 OpenShift CLI (oc)。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

- 要自定义审计日志配置，请输入以下命令：

```
$ oc edit network.operator.openshift.io/cluster
```

提示

您还可以自定义并应用以下 YAML 来配置审计日志记录：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
```

验证

1. **要创建带有网络策略的命名空间，请完成以下步骤：**

   a. **创建命名空间进行验证：**

   ```
   $ cat <<EOF| oc create -f -
   kind: Namespace
   apiVersion: v1
   metadata:
     name: verify-audit-logging
     annotations:
       k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert" }'
   EOF
   ```

   输出示例

   ```
   namespace/verify-audit-logging created
   ```

   b. **为命名空间创建网络策略：**

   ```
   $ cat <<EOF| oc create -n verify-audit-logging -f -
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
     name: deny-all
   spec:
     podSelector:
       matchLabels:
     policyTypes:
     - Ingress
     - Egress
   ---
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
     name: allow-from-same-namespace
   ```

```
  namespace: verify-audit-logging
 spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
   - from:
     - podSelector: {}
  egress:
   - to:
     - namespaceSelector:
       matchLabels:
         kubernetes.io/metadata.name: verify-audit-logging
EOF
```

输出示例

```
networkpolicy.networking.k8s.io/deny-all created
networkpolicy.networking.k8s.io/allow-from-same-namespace created
```

2. 为 default 命名空间中的源流量创建 pod：

```
$ cat <<EOF| oc create -n default -f -
apiVersion: v1
kind: Pod
metadata:
 name: client
spec:
 containers:
  - name: client
    image: registry.access.redhat.com/rhel7/rhel-tools
    command: ["/bin/sh", "-c"]
    args:
     ["sleep inf"]
EOF
```

3. 在 verify-audit-logging 命名空间中创建两个 pod：

```
$ for name in client server; do
cat <<EOF| oc create -n verify-audit-logging -f -
apiVersion: v1
kind: Pod
metadata:
 name: ${name}
spec:
 containers:
  - name: ${name}
    image: registry.access.redhat.com/rhel7/rhel-tools
    command: ["/bin/sh", "-c"]
    args:
     ["sleep inf"]
EOF
done
```

输出示例

```
pod/client created
pod/server created
```

4. 要生成流量并生成网络策略审计日志条目，请完成以下步骤：

   a. 在 verify-audit-logging 命名空间中获取名为 server 的 pod 的 IP 地址：

   ```
   $ POD_IP=$(oc get pods server -n verify-audit-logging -o
   jsonpath='{.status.podIP}')
   ```

   b. 从 default 命名空间中名为 client 的 pod 中 ping 上一个命令的 IP 地址，并确认所有数据包都已丢弃：

   ```
   $ oc exec -it client -n default -- /bin/ping -c 2 $POD_IP
   ```

   输出示例

   ```
   PING 10.128.2.55 (10.128.2.55) 56(84) bytes of data.

   --- 10.128.2.55 ping statistics ---
   2 packets transmitted, 0 received, 100% packet loss, time 2041ms
   ```

   c. 从 verify-audit-logging 命名空间中名为 client 的 pod 中 pingPOD_IP shell 环境变量中保存的 IP 地址，并确认允许所有数据包：

   ```
   $ oc exec -it client -n verify-audit-logging -- /bin/ping -c 2 $POD_IP
   ```

   输出示例

   ```
   PING 10.128.0.86 (10.128.0.86) 56(84) bytes of data.
   64 bytes from 10.128.0.86: icmp_seq=1 ttl=64 time=2.21 ms
   64 bytes from 10.128.0.86: icmp_seq=2 ttl=64 time=0.440 ms

   --- 10.128.0.86 ping statistics ---
   2 packets transmitted, 2 received, 0% packet loss, time 1001ms
   rtt min/avg/max/mdev = 0.440/1.329/2.219/0.890 ms
   ```

5. 显示网络策略审计日志中的最新条目：

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-
headers=true | awk '{ print $1 }') ; do
   oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
 done
```

输出示例

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.
0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_
```

```
dst=8080,tcp_flags=syn
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.
0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_
dst=8080,tcp_flags=syn
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.
0.39,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_
dst=8080,tcp_flags=syn
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert,
direction=from-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert,
direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert,
direction=from-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert,
direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
```

6.4.6. 为命名空间启用出口防火墙和网络策略审计日志

作为集群管理员，您可以为命名空间启用审计日志。

先决条件

- 安装 OpenShift CLI (oc)。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

- 要为命名空间启用审计日志，请输入以下命令：

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/acl-logging='{ "deny": "alert", "allow": "notice" }'
```

其中：

<namespace>
指定命名空间的名称。

提示

您还可以应用以下 YAML 来启用审计日志记录：

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "alert",
        "allow": "notice"
      }
```

输出示例

```
namespace/verify-audit-logging annotated
```

验证

- 显示审计日志中的最新条目：

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-
headers=true | awk '{ print $1 }') ; do
    oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
  done
```

输出示例

```
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert,
direction=from-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert,
direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert,
direction=from-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert,
```

```
direction=to-lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.12
9.2.34,nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,ic
mp_code=0
```

6.4.7. 为命名空间禁用出口防火墙和网络策略审计日志

作为集群管理员，您可以禁用命名空间的审计日志。

先决条件

- 安装 OpenShift CLI (oc) 。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

- 要为命名空间禁用审计日志，请输入以下命令：

```
$ oc annotate --overwrite namespace <namespace> k8s.ovn.org/acl-logging-
```

其中：

<namespace>
指定命名空间的名称。

提示

您还可以应用以下 YAML 来禁用审计日志记录：

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: null
```

输出示例

```
namespace/verify-audit-logging annotated
```

6.4.8. 其他资源

- 关于网络策略
- 为项目配置出口防火墙

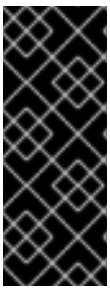
6.5. OPENSHIFT CONTAINER PLATFORM 中的 INGRESS NODE FIREWALL OPERATOR

Ingress Node Firewall Operator 允许管理员在节点级别管理防火墙配置。

6.5.1. Ingress Node Firewall Operator

Ingress Node Firewall Operator 通过将守护进程集部署到您在防火墙配置中指定和管理的节点，在节点级别提供入口防火墙规则。要部署守护进程集，请创建一个 IngressNodeFirewallConfig 自定义资源 (CR)。Operator 应用 IngressNodeFirewallConfig CR 来创建入口节点防火墙守护进程集 daemon，它在与 nodeSelector 匹配的所有节点上运行。

您可以配置 IngressNodeFirewall CR 的规则，并使用 nodeSelector 将值设置为 "true" 的集群。



重要

Ingress Node Firewall Operator 仅支持无状态防火墙规则。

不支持原生 XDP 驱动程序的网络接口控制器 (NIC) 将以较低性能运行。

对于 OpenShift Container Platform 4.14 或更高的版本，您必须在 RHEL 9.0 或更高版本上运行 Ingress Node Firewall Operator。

6.5.2. 安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 OpenShift Container Platform CLI 或 Web 控制台安装 Ingress Node Firewall Operator。

6.5.2.1. 使用 CLI 安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 CLI 安装 Operator。

先决条件

- 已安装 OpenShift CLI（oc）。
- 有管理员特权的帐户。

流程

1. 运行以下命令来创建 openshift-ingress-node-firewall 命名空间：

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
EOF
```

2. 运行以下命令来创建 OperatorGroup CR：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```

```
  name: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. 订阅 Ingress Node Firewall Operator。

   a. 要为 Ingress Node Firewall Operator 创建 **Subscription** CR，请输入以下命令：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ingress-node-firewall-sub
  namespace: openshift-ingress-node-firewall
spec:
  name: ingress-node-firewall
  channel: stable
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. 要验证是否已安装 Operator，请输入以下命令：

```
$ oc get ip -n openshift-ingress-node-firewall
```

   输出示例

```
NAME            CSV                                       APPROVAL    APPROVED
install-5cvnz   ingress-node-firewall.4.16.0-202211122336   Automatic   true
```

5. 要验证 Operator 的版本，请输入以下命令：

```
$ oc get csv -n openshift-ingress-node-firewall
```

   输出示例

```
NAME                                         DISPLAY                       VERSION               REPLACES
PHASE
ingress-node-firewall.4.16.0-202211122336    Ingress Node Firewall Operator    4.16.0-
202211122336    ingress-node-firewall.4.16.0-202211102047    Succeeded
```

6.5.2.2. 使用 Web 控制台安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 Web 控制台安装 Operator。

先决条件

- 已安装 OpenShift CLI（oc）。
- 有管理员特权的帐户。

流程

1. 安装 Ingress Node Firewall Operator：
   a. 在 OpenShift Container Platform Web 控制台中，点击 Operators → OperatorHub。
   b. 从可用的 Operator 列表中选择 Ingress Node Firewall Operator，然后点 Install。
   c. 在 Install Operator 页面中，在 Installed Namespace 下选择 Operator recommended Namespace。
   d. 点 Install。
2. 验证 Ingress Node Firewall Operator 是否已成功安装：
   a. 导航到 Operators → Installed Operators 页面。
   b. 确保 openshift-ingress-node-firewall 项目中列出的 Ingress Node Firewall Operator 的 Status 为 InstallSucceeded。

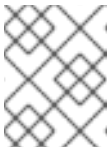
注意

在安装过程中，Operator 可能会显示 Failed 状态。如果安装过程结束后有 InstallSucceeded 信息，您可以忽略这个 Failed 信息。

如果 Operator 没有 InstallSucceeded 状态，请按照以下步骤进行故障排除：

- 检查 Operator Subscriptions 和 Install Plans 选项卡中的 Status 项中是否有任何错误。
- 进入到 Workloads → Pods 页面，在 openshift-ingress-node-firewall 项目中检查 pod 的日志。
- 检查 YAML 文件的命名空间。如果缺少注解，您可以使用以下命令将注解 workload.openshift.io/allowed=management 添加到 Operator 命名空间中：

```
$ oc annotate ns/openshift-ingress-node-firewall
workload.openshift.io/allowed=management
```



注意

对于单节点 OpenShift 集群，openshift-ingress-node-firewall 命名空间需要 workload.openshift.io/allowed=management 注解。

6.5.3. 部署 Ingress Node Firewall Operator

前提条件

- 已安装 Ingress Node Firewall Operator。

流程

要拒绝 Ingress Node Firewall Operator，请创建一个 IngressNodeFirewallConfig 自定义资源，该资源将部署 Operator 的守护进程集。您可以通过应用防火墙规则，将一个或多个 IngressNodeFirewall CRD 部署到节点。

1. 在 openshift-ingress-node-firewall 命名空间中创建 IngressNodeFirewallConfig，名为 ingressnodefirewallconfig。

2. 运行以下命令来部署 Ingress Node Firewall Operator 规则：

```
$ oc apply -f rule.yaml
```

6.5.3.1. Ingress 节点防火墙配置对象

下表中描述了 Ingress Node Firewall 配置对象的字段：

表 6.5. Ingress 节点防火墙配置对象

字段	类型	描述
metadata.name	**字符串**	CR 对象的名称。防火墙规则对象的名称必须是 **ingressnodefirewallconfig**。
metadata.namespace	**字符串**	Ingress Firewall Operator CR 对象的命名空间。**IngressNodeFirewallConfig** CR 必须在 **openshift-ingress-node-firewall** 命名空间中创建。
spec.nodeSelector	**字符串**	通过指定节点标签 (label) 用于目标节点的节点选择约束。例如： `spec:` `  nodeSelector:` `  node-role.kubernetes.io/worker: ""` **注意** **nodeSelector** 中使用的一个标签必须与节点上的标签匹配，以便守护进程集启动。例如，如果节点标签 **node-role.kubernetes.io/worker** 和 **node-type.kubernetes.io/vm** 应用到某个节点，则必须使用 **nodeSelector** 至少设置一个标签设置来使守护进程启动。

注意

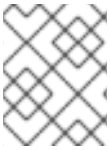
Operator 使用 CR，并在与 **nodeSelector** 匹配的所有节点上创建一个入口节点防火墙守护进程集。

Ingress Node Firewall Operator 示例配置
以下示例中指定了完整的 Ingress Node 防火墙配置：

Ingress 节点防火墙配置对象示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
```

```
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```



注意

Operator 使用 CR，并在与 nodeSelector 匹配的所有节点上创建一个入口节点防火墙守护进程集。

6.5.3.2. Ingress 节点防火墙规则对象

下表中描述了 Ingress Node Firewall 规则对象的字段：

表 6.6. Ingress 节点防火墙规则对象

字段	类型	描述
metadata.name	**字符串**	CR 对象的名称。
interfaces	**array**	此对象的字段指定要应用防火墙规则的接口。例如，**- en0** 和 **- en1**。
nodeSelector	**array**	您可以使用 **nodeSelector** 来选择节点来应用防火墙规则。将指定 **nodeselector** 标签的值设置为 **true** 以应用该规则。
ingress	**object**	**Ingress** 允许您配置允许从外部访问集群中的服务的规则。

Ingress 对象配置
ingress 对象的值在下表中定义：

表 6.7. **Ingress** 对象

字段	类型	描述
sourceCIDRs	**array**	允许您设置 CIDR 块。您可以从不同地址系列配置多个 CIDR。  **注意** 不同的 CIDR 允许您使用相同的顺序规则。如果同一节点有多个 **IngressNodeFirewall** 对象，且带有重叠 CIDR 的接口，则 **order** 字段将指定首先应用该规则。规则以升序应用。

字段	类型	描述
rules	**array**	对于每个 **source.CIDR**，Ingress 防火墙 **rules.order** 对象的顺序以 **1** 开始，每个 CIDR 最多 100 个规则。低顺序规则会首先执行。 **rules.protocolConfig.protocol** 支持以下协议：TCP、UDP、SCTP、ICMP 和 ICMPv6。ICMP 和 ICMPv6 规则可以匹配 ICMP 和 ICMPv6 类型或代码。TCP、UDP 和 SCTP 规则针对单一一个目标端口或一个目标端口范围（格式为 **<start : end-1>**）进行匹配。 将 **rules.action** 设置为 **allow** 以应用规则，**deny** 来禁止规则。  **注意** Ingress 防火墙规则使用阻止任何无效配置的验证 Webhook 进行验证。验证 Webhook 会阻止阻塞任何关键集群服务，如 API 服务器。

Ingress 节点防火墙规则对象示例
以下示例中指定了完整的 Ingress Node 防火墙配置：

Ingress 节点防火墙配置示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall
spec:
  interfaces:
  - eth0
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> 1
  ingress:
  - sourceCIDRs:
      - 172.16.0.0/12
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMP
        icmp:
          icmpType: 8 #ICMP Echo request
      action: Deny
    - order: 20
      protocolConfig:
        protocol: TCP
```

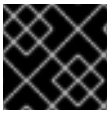
```
      tcp:
        ports: "8000-9000"
    action: Deny
  - sourceCIDRs:
      - fc00:f853:ccd:e793::0/64
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMPv6
        icmpv6:
          icmpType: 128 #ICMPV6 Echo request
      action: Deny
```

1 节点上必须存在 <label_name> 和 <label_value>，且必须与应用到您希望 ingressfirewallconfig CR 运行的节点的 nodeselector 标签和值匹配。<label_value> 可以是 true 或 false。通过使用 nodeSelector 标签，您可以针对单独的节点组为目标，以使用 ingressfirewallconfig CR 应用不同的规则。

零信任 Ingress Node Firewall 规则对象示例

零信任 Ingress 节点防火墙规则可为多接口集群提供额外的安全性。例如，您可以使用零信任 Ingress Node Firewall 规则来丢弃除 SSH 之外的特定接口上的网络流量。

以下示例中指定了零信任 Ingress Node Firewall 规则集的完整配置：



重要

用户需要为其提供应用程序使用的所有端口添加到允许列表，以确保正常工作。

零信任 Ingress 节点防火墙规则示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall-zero-trust
spec:
  interfaces:
  - eth1 1
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> 2
  ingress:
  - sourceCIDRs:
      - 0.0.0.0/0 3
    rules:
    - order: 10
      protocolConfig:
        protocol: TCP
        tcp:
          ports: 22
      action: Allow
    - order: 20
      action: Deny 4
```

1. Network-interface 集群
2. <label_name> 和 <label_value> 需要与应用到 ingressfirewallconfig CR 的特定节点的 nodeSelector 标签和值匹配。
3. 0.0.0.0/0 匹配任何 CIDR
4. action 设置为 Deny

6.5.4. 查看 Ingress Node Firewall Operator 规则

流程

1. 运行以下命令来查看所有当前规则：

```
$ oc get ingressnodefirewall
```

2. 选择返回的 <resource> 名称之一，并运行以下命令来查看规则或配置：

```
$ oc get <resource> <name> -o yaml
```

6.5.5. 对 Ingress Node Firewall Operator 进行故障排除

- 运行以下命令列出已安装的 Ingress Node Firewall 自定义资源定义 (CRD)：

```
$ oc get crds | grep ingressnodefirewall
```

输出示例

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
ingressnodefirewallconfigs.ingressnodefirewall.openshift.io      2022-08-25T10:03:01Z
ingressnodefirewallnodestates.ingressnodefirewall.openshift.io   2022-08-
25T10:03:00Z
ingressnodefirewalls.ingressnodefirewall.openshift.io            2022-08-25T10:03:00Z
```

- 运行以下命令，以查看 Ingress Node Firewall Operator 的状态：

```
$ oc get pods -n openshift-ingress-node-firewall
```

输出示例

```
NAME                                   READY  STATUS       RESTARTS  AGE
ingress-node-firewall-controller-manager  2/2   Running      0         5d21h
ingress-node-firewall-daemon-pqx56        3/3   Running      0         5d21h
```

以下字段提供有关 Operator 状态的信息：READY、STATUS、AGE、和 RESTARTS。当 Ingress Node Firewall Operator 将守护进程集部署到分配的节点时，STATUS 字段为 Running。

- 运行以下命令来收集所有入口防火墙节点 pod 的日志：

```
$ oc adm must-gather – gather_ingress_node_firewall
```

在 sos 节点的报告中，其中包含位于 /sos_commands/ebpf 的 eBPF bpftool 输出的报告。这些报告包括用于或作为入口防火墙 XDP 处理数据包处理、更新统计信息和发出事件的查找表。

6.6. EGRESS 防火墙

6.6.1. 查看项目的出口防火墙

作为集群管理员，您可以列出任何现有出口防火墙的名称，并查看特定出口防火墙的流量规则。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

6.6.1.1. 查看 EgressFirewall 对象

您可以查看集群中的 EgressFirewall 对象。

先决条件

- 使用 OVN-Kubernetes 网络插件的集群。
- 安装 OpenShift 命令行界面 (CLI)，通常称为 oc。
- 您必须登录集群。

流程

1. 可选：要查看集群中定义的 EgressFirewall 对象的名称，请输入以下命令：

```
$ oc get egressfirewall --all-namespaces
```

2. 要检查策略，请输入以下命令。将 <policy_name> 替换为要检查的策略名称。

```
$ oc describe egressfirewall <policy_name>
```

输出示例

```
Name:  default
Namespace: project1
Created: 20 minutes ago
Labels:  <none>
Annotations: <none>
Rule:  Allow to 1.2.3.0/24
Rule:  Allow to www.example.com
Rule:  Deny to 0.0.0.0/0
```

6.6.2. 为项目编辑出口防火墙

作为集群管理员，您可以修改现有出口防火墙的网络流量规则。

6.6.2.1. 编辑 EgressFirewall 对象

作为集群管理员，您可以更新一个项目的出口防火墙。

先决条件

- 使用 OVN-Kubernetes 网络插件的集群。
- 安装 OpenShift CLI (oc) 。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 查找项目的 EgressFirewall 对象的名称。将 <project> 替换为项目的名称。

```
$ oc get -n <project> egressfirewall
```

2. 可选：如果您在创建出口网络防火墙时没有保存 EgressFirewall 对象的副本，请输入以下命令来创建副本。

```
$ oc get -n <project> egressfirewall <name> -o yaml > <filename>.yaml
```

将 <project> 替换为项目的名称。将 <name> 替换为 Pod 的名称。将 <filename> 替换为要将 YAML 保存到的文件的名称。

3. 修改策略规则后，请输入以下命令替换 EgressFirewall 对象。将 <filename> 替换为包含更新的 EgressFirewall 对象的文件名称。

```
$ oc replace -f <filename>.yaml
```

6.6.3. 从项目中删除出口防火墙

作为集群管理员，您可以从项目中删除出口防火墙，从而删除对项目的离开 OpenShift Container Platform 集群的网络流量的限制。

6.6.3.1. 删除 EgressFirewall 对象

作为集群管理员，您可以从项目中删除出口防火墙。

先决条件

- 使用 OVN-Kubernetes 网络插件的集群。
- 安装 OpenShift CLI (oc) 。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 查找项目的 EgressFirewall 对象的名称。将 <project> 替换为项目的名称。

```
$ oc get -n <project> egressfirewall
```

2. 输入以下命令删除 EgressFirewall 对象。将 <project> 替换为项目名称，<name> 替换为对象名称。

```
$ oc delete -n <project> egressfirewall <name>
```

6.6.4. 为项目配置出口防火墙

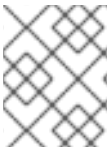
作为集群管理员，您可以为项目创建一个出口防火墙，用于限制离开 OpenShift Container Platform 集群的出口流量。

6.6.4.1. 出口防火墙在一个项目中的工作原理

作为集群管理员，您可以使用一个*出口防火墙*来限制集群内的一些 pod 或所有 pod 可以访问的外部主机。出口防火墙适用于以下情况：

- pod 只能连接到内部主机，且无法启动到公共互联网的连接。
- pod 只能连接到公共互联网，且无法启动到 OpenShift Container Platform 集群以外的内部主机的连接。
- pod 无法访问 OpenShift Container Platform 集群外的特定内部子网或主机。
- pod 只能连接到特定的外部主机。

例如，您可以允许某一个项目访问指定的 IP 范围，但拒绝其他项目对同一 IP 范围的访问。或者您可以限制应用程序开发人员从 Python pip 的镜像点进行更新，并强制要求更新只能来自于批准的源。



注意

出口防火墙不适用于主机网络命名空间。启用主机网络的 Pod 不受出口防火墙规则的影响。

您可以通过创建一个 EgressFirewall 自定义资源（CR）对象来配置出口防火墙策略。出口防火墙与满足以下任一条件的网络流量匹配：

- CIDR 格式的 IP 地址范围
- 解析为 IP 地址的 DNS 名称
- 端口号
- 协议是以下协议之一： TCP、UDP 和 SCTP

重要

如果您的出口防火墙包含 0.0.0.0/0 的拒绝规则，则阻止访问 OpenShift Container Platform API 服务器。您必须为每个 IP 地址添加允许规则，或使用出口策略规则中的 nodeSelector 类型允许规则来连接到 API 服务器。

以下示例演示了确保 API 服务器访问所需的出口防火墙规则的顺序：

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
  namespace: <namespace> 1
spec:
  egress:
  - to:
      cidrSelector: <api_server_address_range> 2
    type: Allow
# ...
  - to:
      cidrSelector: 0.0.0.0/0 3
    type: Deny
```

1. 出口防火墙的命名空间。
2. 包含 OpenShift Container Platform API 服务器的 IP 地址范围。
3. 一个全局拒绝规则会阻止访问 OpenShift Container Platform API 服务器。

要查找 API 服务器的 IP 地址，请运行 oc get ep kubernetes -n default。

如需更多信息，请参阅 BZ#1988324。

警告

出口防火墙规则不适用于通过路由器的网络流量。任何有权创建 Route CR 对象的用户，都可以通过创建指向禁止的目的地的路由来绕过出口防火墙策略规则。

6.6.4.1.1. 出口防火墙的限制

出口防火墙有以下限制：

- 项目不能有一个以上的 EgressFirewall 对象。
- 每个项目最多可定义一个具有最多 8,000 个规则的 EgressFirewall 对象。
- 如果您在 Red Hat OpenShift Networking 中使用带有共享网关模式的 OVN-Kubernetes 网络插件，则返回入口回复会受到出口防火墙规则的影响。如果出口防火墙规则丢弃入口回复目的地 IP，流量将被丢弃。

违反这些限制会导致项目的出口防火墙出现问题。因此，所有外部网络流量都会被丢弃，这可能会给您的组织造成安全风险。

可在 kube-node-lease、kube-public、kube-system、openshift 和 openshift- 项目中创建一个 Egress Firewall 资源。

6.6.4.1.2. 出口防火墙策略规则的匹配顺序

出口防火墙策略规则按照它们定义的顺序来评估，从第一个到最后一个的顺序。第一个与 pod 的出口连接匹配的规则会被应用。该连接会忽略后续的所有规则。

6.6.4.1.3. 域名服务器 (DNS) 解析如何工作

如果您在 egress 防火墙策略规则中使用 DNS 名称，则正确解析域名会受到以下限制：

- 域名更新会根据生存时间（TTL）持续时间进行轮询。默认情况下，持续时间为 30 分钟。当出口防火墙控制器查询本地名称服务器以获取域名时，如果响应包含 TTL 且 TTL 小于 30 分钟，控制器会将该 DNS 名称的持续时间设置为返回的值。每个 DNS 名称都会在 DNS 记录的 TTL 过期后查询。
- 在需要时，pod 必须通过相同的本地名称服务器解析域名。否则，egress 防火墙控制器和 pod 已知的域的 IP 地址可能会有所不同。如果主机名的 IP 地址不同，则出口防火墙的强制实施可能不一致。
- 因为出口防火墙控制器和 pod 异步轮询相同的本地名称服务器，所以 pod 可能会在出口控制器执行前获取更新的 IP 地址，从而导致竞争条件。由于这个限制，仅建议在 EgressFirewall 对象中使用域名来更改 IP 地址的域。

注意

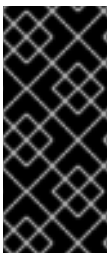
在出口防火墙策略中使用 DNS 名称不会影响通过 CoreDNS 的本地 DNS 解析。

但是，如果您的出口防火墙策略使用域名，并且外部 DNS 服务器处理受影响 pod 的 DNS 解析，则必须包括允许访问 DNS 服务器的 IP 地址的出口防火墙规则。

6.6.4.1.3.1. 改进了 DNS 解析并解析通配符域名

在某些情况下，与 DNS 记录关联的 IP 地址会频繁更改，或者您可能希望在出口（egress）防火墙策略规则中指定通配符域名。

在这种情况下，OVN-Kubernetes 集群管理器为每个出口防火墙策略规则中使用的每个唯一 DNS 名称创建一个 DNSNameResolver 自定义资源对象。此自定义资源存储以下信息：



重要

改进了出口防火墙规则的 DNS 解析只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

DNSNameResolver CR 定义示例

```
apiVersion: networking.openshift.io/v1alpha1
kind: DNSNameResolver
spec:
  name: www.example.com. 1
```

```
status:
  resolvedNames:
  - dnsName: www.example.com. 2
    resolvedAddress:
    - ip: "1.2.3.4" 3
      ttlSeconds: 60 4
      lastLookupTime: "2023-08-08T15:07:04Z" 5
```

1. DNS 名称。这可以是标准 DNS 名称或通配符 DNS 名称。对于通配符 DNS 名称，DNS 名称解析信息包含与通配符 DNS 名称匹配的所有 DNS 名称。
2. 与 spec.name 字段匹配的已解析 DNS 名称。如果spec.name 字段包含通配符 DNS 名称，则会创建多个 dnsName 条目，其中包含在解析时与通配符 DNS 名称匹配的标准 DNS 名称。如果通配符 DNS 名称也可以成功解析，则此字段也会存储通配符 DNS 名称。
3. 与 DNS 名称关联的当前 IP 地址。
4. 最后一次生存时间 (TTL) 持续时间。
5. 最后查找时间。

如果在 DNS 解析 DNS 名称的过程中，查询中的 DNS 名称与 DNSNameResolver CR 中定义的任何名称匹配，那么在 CR status 字段中会相应地更新之前的信息。如果 DNS 通配符名称查找失败，会在默认的 TTL 为 30 分钟后重试请求。

OVN-Kubernetes 集群管理器监视对 EgressFirewall 自定义资源对象的更新，并在更新发生时创建、修改或删除与这些出口防火墙策略关联的 DNSNameResolver CR。

警告

不要直接修改 DNSNameResolver 自定义资源。这可能导致出口防火墙的不需要的行为。

6.6.4.2. EgressFirewall 自定义资源（CR）对象

您可以为出口防火墙定义一个或多个规则。规则是一个 Allow 规则，也可以是一个 Deny 规则，它包括规则适用的流量规格。

以下 YAML 描述了 EgressFirewall CR 对象：

EgressFirewall 对象

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1. 对象的名称必须是 default。
2. 以下部分所述，一个或多个出口网络策略规则的集合。

6.6.4.2.1. EgressFirewall 规则

以下 YAML 描述了一个出口防火墙规则对象。用户可以选择 CIDR 格式的 IP 地址范围、域名，或使用 nodeSelector 允许或拒绝出口流量。egress 小节需要一个包括一个或多个对象的数组。

出口策略规则小节

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns_name> 4
    nodeSelector: <label_name>: <label_value> 5
  ports: 6
     ...
```

1. 规则类型。该值必须是 Allow 或 Deny。
2. 描述出口流量匹配规则的小节，该规则指定 cidrSelector 字段或 dnsName 字段。您不能在同一规则中使用这两个字段。
3. CIDR 格式的 IP 地址范围。
4. DNS 域名。
5. 标签是用户定义的键/值对。标签附加到对象，如 pod。nodeSelector 允许选择一个或多个节点标签，并附加到 pod。
6. 可选：描述该规则的网络端口和协议集合的小节。

端口小节

```
ports:
- port: <port> 1
  protocol: <protocol> 2
```

1. 网络端口，比如 80 或 443。如果为这个字段指定值，还必须为 protocol 指定一个值。
2. 网络协议。该值必须是 TCP、UDP 或 SCTP。

6.6.4.2.2. EgressFirewall CR 对象示例

以下示例定义了几个出口防火墙策略规则：

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
```

```
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 出口防火墙策略规则对象的集合。

以下示例定义了一个策略规则，即如果流量使用 TCP 协议和目标端口 80，或任何协议和目标端口 443，则拒绝通过 172.16.1.1 IP 地址到主机的流量。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
  - type: Deny
    to:
      cidrSelector: 172.16.1.1
    ports:
    - port: 80
      protocol: TCP
    - port: 443
```

6.6.4.2.3. EgressFirewall 的 nodeSelector 示例

作为一个集群管理员，您可以使用 nodeSelector 指定标签来允许或拒绝集群中节点的出口流量。标签可以应用到一个或多个节点。以下是带有 region=east 标签的示例：

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
    egress:
    - to:
        nodeSelector:
          matchLabels:
            region: east
      type: Allow
```

提示

除了为每个节点 IP 地址添加手动规则外，使用节点选择器创建标签，允许出口防火墙后面的 pod 访问主机网络 pod。

6.6.4.3. 创建出口防火墙策略对象

作为集群管理员，您可以为项目创建一个出口防火墙策略对象。

重要

如果项目已经定义了 EgressFirewall 对象，您必须编辑现有策略来更改出口防火墙规则。

先决条件

- 使用 OVN-Kubernetes 网络插件的集群。
- 安装 OpenShift CLI（oc）。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 创建策略规则：
   a. 创建一个 <policy_name>.yaml 文件，其中 <policy_name> 描述出口策略规则。
   b. 在您创建的文件中，定义出口策略对象。
2. 运行以下命令来创建策略对象。将 <policy_name> 替换为策略的名称， <project> 替换为规则应用到的项目。

```
$ oc create -f <policy_name>.yaml -n <project>
```

   在以下示例中，在名为 project1 的项目中创建一个新的 EgressFirewall 对象：

```
$ oc create -f default.yaml -n project1
```

   输出示例

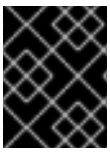
```
egressfirewall.k8s.ovn.org/v1 created
```

3. 可选：保存 <policy_name>.yaml 文件，以便在以后进行修改。

6.7. 配置 IPSEC 加密

启用 IPsec 后，您可以加密节点之间的内部 pod 到 pod 集群流量，以及集群外部的 pod 和 IPsec 端点之间的外部流量。OVN-Kubernetes 集群网络上的节点之间的所有 pod 到 pod 网络流量都使用 IPsec *传输模式* 加密。

默认禁用 IPsec。它可以在安装集群期间或之后启用。有关集群安装的详情，请参阅 OpenShift Container Platform 安装概述。



重要

如果您的集群在 Red Hat OpenShift Container Platform 中使用 托管的 control plane，则 IPsec 不支持 pod 到 pod，以及到外部主机的 IPsec 加密。

注意

IBM Cloud® 上的 IPsec 仅支持 NAT-T。不支持使用 ESP。

使用以下文档中的流程来：

- 在集群安装后启用和禁用 IPSec
- 为集群和外部主机之间的流量配置 IPsec 加密
- 验证 IPsec 是否加密不同节点上的 pod 之间的流量

6.7.1. 操作模式

在 OpenShift Container Platform 集群中使用 IPsec 时，您可以从以下操作模式中选择：

表 6.8. 操作的 IPsec 模式

模式	描述	default
Disabled	没有流量被加密。这是集群的默认设置。	是
Full	pod 到 pod 的流量被加密，如"由 pod 到 pod IPsec 加密的网络流量类型"中所述。完成 IPsec 所需的配置步骤后，可以加密到外部节点的流量。	否
外部	完成 IPsec 所需的配置步骤后，可以加密到外部节点的流量。	否

6.7.2. 先决条件

对于将流量加密到外部主机的 IPsec 支持，请确保满足以下先决条件：

- OVN-Kubernetes 网络插件必须配置为本地网关模式，其中 **ovnKubernetesConfig.gatewayConfig.routingViaHost=true**。
- 已安装 NMState Operator。指定 IPsec 配置需要这个 Operator。如需更多信息，请参阅关于 Kubernetes NMState Operator。

  注意

  NMState Operator 仅在 Google Cloud Platform (GCP) 上支持来配置 IPsec。

- 已安装 Butane 工具 (**butane**)。要安装 Butane，请参阅安装 Butane。

这些先决条件需要将证书添加到主机 NSS 数据库中，并配置 IPsec 与外部主机进行通信。

6.7.3. 启用 IPsec 时的网络连接要求

您必须配置机器之间的网络连接，以允许 OpenShift Container Platform 集群组件进行通信。每台机器都必须能够解析集群中所有其他机器的主机名。

表 6.9. 用于全机器到所有机器通信的端口

协议	port	描述
UDP	**500**	IPsec IKE 数据包
	4500	IPsec NAT-T 数据包
ESP	N/A	IPsec Encapsulating Security Payload(ESP)

6.7.4. pod 到 pod 流量的 IPsec 加密

对于 pod 到 pod 流量的 IPsec 加密，以下小节描述了加密哪些特定的 pod 到 pod 的流量，使用什么加密协议，以及如何处理 X.509 证书。这些部分不适用于集群和外部主机之间的 IPsec 加密，您必须为特定的外部网络基础架构手动配置。

6.7.4.1. 由 pod 到 pod IPsec 加密的网络流量类型

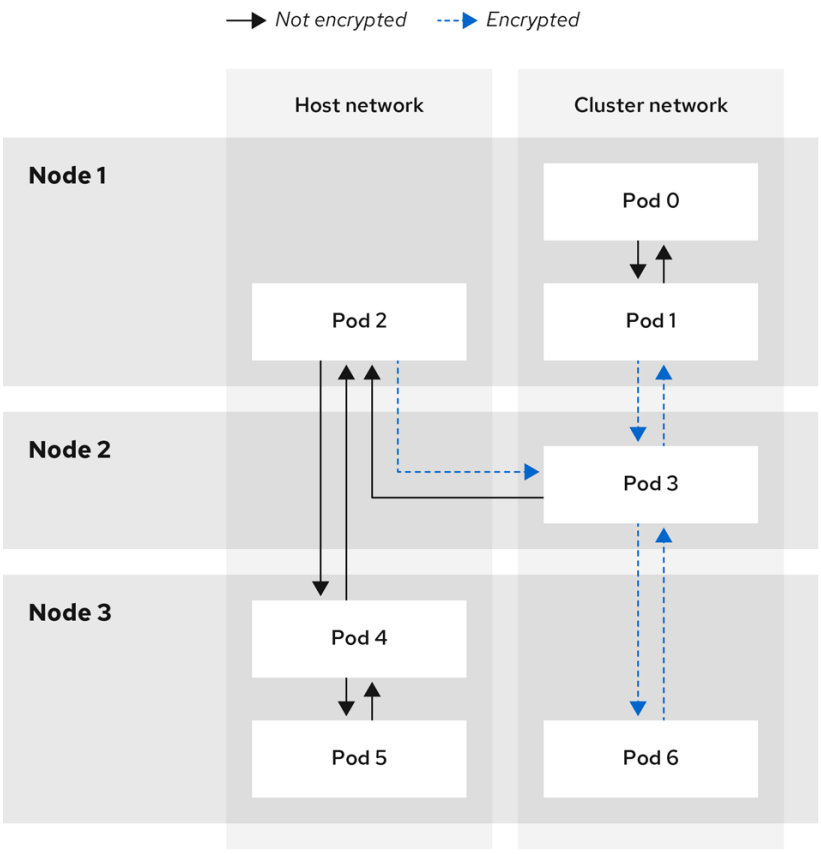
启用 IPsec 后，只有 pod 间的以下网络流量会被加密：

- 集群网络的不同节点上的 pod 间的流量
- 从主机网络上的 pod 流量到集群网络上的 pod

以下流量流没有加密：

- 集群网络上同一节点上的 pod 间的流量
- 主机网络上的 pod 间的流量
- 从集群网络上的 pod 流量到主机网络上的 pod

下图中显示了加密和未加密的流程：



6.7.4.2. 加密协议和 IPsec 模式

使用的加密机制是 AES-GCM-16-256。完整性检查值（ICV）为 16 字节。密钥长度为 256 位。

使用的 IPsec 模式是 *传输模式*，这是通过向原始数据包的 IP 标头添加封装安全 Payload (ESP) 标头来加密端到端通信的模式。OpenShift Container Platform 目前不支持 pod 到 pod 通信的 IPsec *Tunnel 模式*。

6.7.4.3. 安全证书生成和轮转

Cluster Network Operator（CNO）生成自签名 X.509 证书颁发机构（CA），该颁发机构（CA）用于加密。来自每个节点的证书签名请求（CSR）由 CNO 自动实现。

CA 的有效期为 10 年。独立节点证书的有效期为 5 年，并在 4 年半后自动轮转。

6.7.5. 外部流量的 IPsec 加密

OpenShift Container Platform 支持到具有必须提供的 TLS 证书的外部主机流量的 IPsec 加密。

6.7.5.1. 支持的平台

在以下平台上支持此功能：

- 裸机
- Google Cloud Platform (GCP)
- Red Hat OpenStack Platform(RHOSP)

- VMware vSphere

重要

如果您有 Red Hat Enterprise Linux (RHEL) worker 节点，它们不支持外部流量的 IPsec 加密。

如果您的集群在 Red Hat OpenShift Container Platform 中使用托管的 control plane，则不支持将流量加密到外部主机的 IPsec。

6.7.5.2. 限制

确保观察到以下限制：

- 在为外部流量配置 IPsec 时，NMState Operator 目前不支持 IPv6 配置。
- 提供的证书捆绑包中的证书通用名称(CN)不能以 ovs_ 前缀开头，因为这个命名可以与每个节点的网络安全服务(NSS)数据库中的 pod 到 pod 的 IPsec CN 名称冲突。

6.7.6. 启用 IPsec 加密

作为集群管理员，您可以在集群和外部 IPsec 端点之间，启用 pod 到 pod IPsec 加密和 IPsec 加密。

您可以使用以下模式之一配置 IPsec：

- **Full** ：pod 到 pod 和外部流量的加密
- **External** ：对外部流量进行加密

如果您需要除 pod 到 pod 流量加密外，还需要对外部流量配置加密，则需要完成"为外部流量配置 IPsec 加密"流程。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录集群。
- 您已将集群 MTU 的大小减少为 46 字节，以允许 IPsec ESP 标头的开销。

流程

1. 要启用 IPsec 加密，请输入以下命令：

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "ipsecConfig":{
          "mode":<mode>
        }}}}}'
```

其中：

模式

指定 External 以加密到外部主机的流量，或者指定 Full 来加密 pod 到 pod 流量，以及可选的到外部主机的流量。默认情况下禁用 IPsec。

2. 可选： 如果您需要加密到外部主机的流量，请完成"为外部流量配置 IPsec 加密"流程。

验证

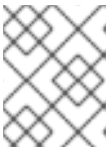
1. 要查找 OVN-Kubernetes data plane pod 的名称，请输入以下命令：

```
$ oc get pods -n openshift-ovn-kubernetes -l=app=ovnkube-node
```

输出示例

```
ovnkube-node-5xqbf                 8/8     Running   0          28m
ovnkube-node-6mwcx                 8/8     Running   0          29m
ovnkube-node-ck5fr                 8/8     Running   0          31m
ovnkube-node-fr4ld                 8/8     Running   0          26m
ovnkube-node-wgs4l                 8/8     Running   0          33m
ovnkube-node-zfvcl                 8/8     Running   0          34m
```

2. 运行以下命令，验证集群中是否启用了 IPsec：



注意

作为集群管理员，您可以在 IPsec 配置为 Full 模式时验证集群中 pod 之间是否启用了 IPsec。此步骤不会验证 IPsec 是否在集群和外部主机间工作。

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-node-<XXXXX> ovn-nbctl --no-leader-only get nb_global . ipsec
```

其中：

<XXXXX>

指定上一步中 pod 的随机字符序列。

输出示例

```
true
```

6.7.7. 为外部流量配置 IPsec 加密

作为集群管理员，要使用 IPsec 加密外部流量，您必须为网络基础架构配置 IPsec，包括提供 PKCS#12 证书。因为这个流程使用 Butane 创建机器配置，所以必须安装 butane 命令。

注意

应用机器配置后，Machine Config Operator 会重新引导集群中受影响的节点，以推出新机器配置。

先决条件

- 安装 OpenShift CLI（oc）。
- 您已在本地计算机上安装了 butane 工具。
- 在集群中安装了 NMState Operator。
- 以具有 cluster-admin 权限的用户身份登录集群。
- 您有一个 IPsec 端点的现存的 PKCS#12 证书，以及一个 PEM 格式的 CA 证书。
- 您在集群中的 Full 或 External 模式中启用了 IPsec。
- OVN-Kubernetes 网络插件必须配置为本地网关模式，其中 ovnKubernetesConfig.gatewayConfig.routingViaHost=true。

流程

1. 使用 NMState Operator 节点网络配置策略创建 IPsec 配置。如需更多信息，请参阅 Libreswan 作为 IPsec VPN 实现。

   a. 要识别 IPsec 端点的集群节点的 IP 地址，请输入以下命令：

   ```
   $ oc get nodes
   ```

   b. 创建名为 ipsec-config.yaml 的文件，其中包含 NMState Operator 的节点网络配置策略，如下例所示。有关 NodeNetworkConfigurationPolicy 对象的概述，请参阅 Kubernetes NMState 项目。

   NMState IPsec 传输配置示例

   ```
   apiVersion: nmstate.io/v1
   kind: NodeNetworkConfigurationPolicy
   metadata:
     name: ipsec-config
   spec:
     nodeSelector:
       kubernetes.io/hostname: "<hostname>" (1)
     desiredState:
       interfaces:
       - name: <interface_name> (2)
         type: ipsec
         libreswan:
           left: <cluster_node> (3)
           leftid: '%fromcert'
           leftrsasigkey: '%cert'
           leftcert: left_server
           leftmodecfgclient: false
           right: <external_host> (4)
           rightid: '%fromcert'
           rightrsasigkey: '%cert'
           rightsubnet: <external_address>/32 (5)
           ikev2: insist
           type: transport
   ```

   (1) 指定要将策略应用到的主机名。此主机充当 IPsec 配置中的左侧主机。

2 指定要在主机上创建的接口名称。

3 指定在集群端终止 IPsec 隧道的集群节点的主机名。名称应匹配您提供的 PKCS#12 证书中的 SAN [Subject Alternate Name]。

4 指定外部主机名，如 host.example.com。名称应与您提供的 PKCS#12 证书中的 SAN [Subject Alternate Name] 匹配。

5 指定外部主机的 IP 地址，如 10.1.2.3/32。

NMState IPsec 隧道配置示例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ipsec-config
spec:
  nodeSelector:
    kubernetes.io/hostname: "<hostname>" 1
  desiredState:
    interfaces:
    - name: <interface_name> 2
      type: ipsec
      libreswan:
        left: <cluster_node> 3
        leftid: '%fromcert'
        leftmodecfgclient: false
        leftrsasigkey: '%cert'
        leftcert: left_server
        right: <external_host> 4
        rightid: '%fromcert'
        rightrsasigkey: '%cert'
        rightsubnet: <external_address>/32 5
        ikev2: insist
        type: tunnel
```

1 指定要将策略应用到的主机名。此主机充当 IPsec 配置中的左侧主机。

2 指定要在主机上创建的接口名称。

3 指定在集群端终止 IPsec 隧道的集群节点的主机名。名称应匹配您提供的 PKCS#12 证书中的 SAN [Subject Alternate Name]。

4 指定外部主机名，如 host.example.com。名称应与您提供的 PKCS#12 证书中的 SAN [Subject Alternate Name] 匹配。

5 指定外部主机的 IP 地址，如 10.1.2.3/32。

c. 要配置 IPsec 接口，请输入以下命令：

```
$ oc create -f ipsec-config.yaml
```

2. 提供以下证书文件以添加到每个主机上的网络安全服务(NSS)数据库中。这些文件在后续步骤中作为 Butane 配置的一部分导入。

   - left_server.p12：IPsec 端点的证书捆绑包
   - ca.pem ：您使用签名证书的证书颁发机构

3. 创建机器配置以将证书添加到集群中：

   a. 要为 control plane 和 worker 节点创建 Butane 配置文件，请输入以下命令：

```
$ for role in master worker; do
  cat >> "99-ipsec-${role}-endpoint-config.bu" <<-EOF
  variant: openshift
  version: 4.16.0
  metadata:
    name: 99-${role}-import-certs
    labels:
      machineconfiguration.openshift.io/role: $role
  systemd:
    units:
    - name: ipsec-import.service
      enabled: true
      contents: |
        [Unit]
        Description=Import external certs into ipsec NSS
        Before=ipsec.service

        [Service]
        Type=oneshot
        ExecStart=/usr/local/bin/ipsec-addcert.sh
        RemainAfterExit=false
        StandardOutput=journal

        [Install]
        WantedBy=multi-user.target
  storage:
    files:
    - path: /etc/pki/certs/ca.pem
      mode: 0400
      overwrite: true
      contents:
        local: ca.pem
    - path: /etc/pki/certs/left_server.p12
      mode: 0400
      overwrite: true
      contents:
        local: left_server.p12
    - path: /usr/local/bin/ipsec-addcert.sh
      mode: 0740
      overwrite: true
      contents:
        inline: |
          #!/bin/bash -e
          echo "importing cert to NSS"
          certutil -A -n "CA" -t "CT,C,C" -d /var/lib/ipsec/nss/ -i /etc/pki/certs/ca.pem
          pk12util -W "" -i /etc/pki/certs/left_server.p12 -d /var/lib/ipsec/nss/
```

```
        certutil -M -n "left_server" -t "u,u,u" -d /var/lib/ipsec/nss/
EOF
done
```

b. 要将上一步中创建的 Butane 文件转换为机器配置，请输入以下命令：

```
$ for role in master worker; do
  butane 99-ipsec-${role}-endpoint-config.bu -o ./99-ipsec-$role-endpoint-
config.yaml
done
```

4. 要将机器配置应用到集群，请输入以下命令：

```
$ for role in master worker; do
  oc apply -f 99-ipsec-${role}-endpoint-config.yaml
done
```

重要

当 Machine Config Operator (MCO) 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都在外部 IPsec 连接可用前更新。

5. 输入以下命令检查机器配置池状态：

```
$ oc get mcp
```

成功更新的节点具有以下状态：UPDATED=true、UPDATING=false、DEGRADED=false。

注意

默认情况下，MCO 会一次在一个池中更新一个机器，从而导致迁移总时间随着集群大小的增加而增加。

6. 要确认 IPsec 机器配置已被成功推出，请输入以下命令：

a. 确认已创建了 IPsec 机器配置：

```
$ oc get mc | grep ipsec
```

输出示例

```
80-ipsec-master-extensions      3.2.0      6d15h
80-ipsec-worker-extensions      3.2.0      6d15h
```

b. 确认 IPsec 扩展应用到 control plane 节点：

```
$ oc get mcp master -o yaml | grep 80-ipsec-master-extensions -c
```

预期输出

```
2
```

c. 确认 IPsec 扩展已应用到 worker 节点：

```
$ oc get mcp worker -o yaml | grep 80-ipsec-worker-extensions -c
```

预期输出

```
2
```

其他资源

- 有关 nmstate IPsec API 的更多信息，请参阅 IPsec 加密

6.7.8. 为外部 IPsec 端点禁用 IPsec 加密

作为集群管理员，您可以删除外部主机的现有 IPsec 隧道。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录集群。
- 您在集群中的 Full 或 External 模式中启用了 IPsec。

流程

1. 使用以下 YAML 创建名为 remove-ipsec-tunnel.yaml 的文件：

```
kind: NodeNetworkConfigurationPolicy
apiVersion: nmstate.io/v1
metadata:
  name: <name>
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  desiredState:
    interfaces:
    - name: <tunnel_name>
      type: ipsec
      state: absent
```

其中：

name
: 指定节点网络配置策略的名称。

node_name
: 指定您要删除的 IPsec 隧道的节点名称。

tunnel_name
: 指定现有 IPsec 隧道的接口名称。

2. 要删除 IPsec 隧道，请输入以下命令：

```
$ oc apply -f remove-ipsec-tunnel.yaml
```

6.7.9. 禁用 IPsec 加密

作为集群管理员，您可以禁用 IPsec 加密。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 要禁用 IPsec 加密，请输入以下命令：

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "ipsecConfig":{
          "mode":"Disabled"
        }}}}}'
```

2. 可选：您可以将集群 MTU 的大小增加 46 个字节，因为 IP 数据包中不再有 IPsec ESP 标头的开销。

6.7.10. 其他资源

- 在 Red Hat Enterprise Linux (RHEL) 9 中配置使用 IPsec 的 VPN
- 安装 Butane
- 关于 OVN-Kubernetes Container Network Interface (CNI) 网络插件
- 更改集群网络的 MTU
- Network [operator.openshift.io/v1] API

第 7 章 OPENSHIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR

Cluster Network Operator (CNO)在 OpenShift Container Platform 集群上部署和管理集群网络组件，包括在安装过程中为集群选择的 Container Network Interface (CNI) 网络插件。

7.1. CLUSTER NETWORK OPERATOR

Cluster Network Operator 从 operator.openshift.io API 组实现 network API。Operator 通过使用守护进程集部署 OVN-Kubernetes 网络插件，或部署您在集群安装过程中选择的网络供应商插件。

流程

Cluster Network Operator 在安装过程中被部署为一个 Kubernetes 部署。

1. 运行以下命令，以查看部署状态：

```
$ oc get -n openshift-network-operator deployment/network-operator
```

输出示例

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
network-operator   1/1     1            1           56m
```

2. 运行以下命令，以查看 Cluster Network Operator 的状态：

```
$ oc get clusteroperator/network
```

输出示例

```
NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.5.4     True        False         False      50m
```

以下字段提供有关 Operator 状态的信息：AVAILABLE、Progressing 和 DEGRADED。当 Cluster Network Operator 报告可用状态条件时，AVAILABLE 字段为 True。

7.2. 查看集群网络配置

每个 OpenShift Container Platform 新安装都有一个名为 cluster 的 network.config 对象。

流程

- 使用 oc describe 命令查看集群网络配置：

```
$ oc describe network.config/cluster
```

输出示例

```
Name:         cluster
Namespace:
Labels:       <none>
```

```
Annotations:  <none>
API Version:  config.openshift.io/v1
Kind:         Network
Metadata:
  Self Link:          /apis/config.openshift.io/v1/networks/cluster
Spec: 1
  Cluster Network:
    Cidr:         10.128.0.0/14
    Host Prefix:  23
  Network Type:   OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: 2
  Cluster Network:
    Cidr:               10.128.0.0/14
    Host Prefix:        23
  Cluster Network MTU:  8951
  Network Type:         OpenShiftSDN
  Service Network:
    172.30.0.0/16
Events:  <none>
```

1 Spec 字段显示集群网络的已配置状态。

2 Status 字段显示集群网络配置的当前状态。

7.3. 查看 CLUSTER NETWORK OPERATOR 状态

您可以使用 oc describe 命令来检查状态并查看 Cluster Network Operator 的详情。

流程

- 运行以下命令，以查看 Cluster Network Operator 的状态：

```
$ oc describe clusteroperators/network
```

7.4. 查看 CLUSTER NETWORK OPERATOR 日志

您可以使用 oc logs 命令来查看 Cluster Network Operator 日志。

流程

- 运行以下命令，以查看 Cluster Network Operator 的日志：

```
$ oc logs --namespace=openshift-network-operator deployment/network-operator
```

7.5. CLUSTER NETWORK OPERATOR 配置

集群网络的配置作为 Cluster Network Operator(CNO)配置的一部分指定，并存储在名为 cluster 的自定义资源(CR)对象中。CR 指定 operator.openshift.io API 组中的 Network API 的字段。

CNO 配置在集群安装过程中从 Network.config.openshift.io API 组中的 Network API 继承以下字段：

clusterNetwork

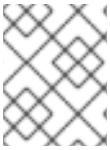
从中分配 Pod IP 地址的 IP 地址池。

serviceNetwork

服务的 IP 地址池.

defaultNetwork.type

集群网络插件。OVNKubernetes 是安装期间唯一支持的插件。



注意

在集群安装后，您只能修改 clusterNetwork IP 地址范围。默认网络类型只能通过迁移从 OpenShift SDN 改为 OVN-Kubernetes。

您可以通过在名为 **cluster** 的 CNO 对象中设置**defaultNetwork** 对象的字段来为集群指定集群网络插件配置。

7.5.1. Cluster Network Operator 配置对象

下表中描述了 Cluster Network Operator(CNO)的字段：

表 7.1. Cluster Network Operator 配置对象

字段	类型	描述
metadata.name	**字符串**	CNO 对象的名称。这个名称始终是 **集群**。
spec.clusterNetwork	**array**	用于指定从哪些 IP 地址块分配 Pod IP 地址以及集群中每个节点的子网前缀长度的列表。例如： `spec:` `  clusterNetwork:` `  - cidr: 10.128.0.0/19` `  hostPrefix: 23` `  - cidr: 10.128.32.0/19` `  hostPrefix: 23`
spec.serviceNetwork	**array**	服务的 IP 地址块。OpenShift SDN 和 OVN-Kubernetes 网络插件只支持服务网络的一个 IP 地址块。例如： `spec:` `  serviceNetwork:` `  - 172.30.0.0/14` 此值是只读的，在集群安装过程中从名为 **cluster** 的 **Network.config.openshift.io** 对象继承。
spec.defaultNetwork	**object**	为集群网络配置网络插件。
spec.kubeProxyConfig	**object**	此对象的字段指定 kube-proxy 配置。如果使用 OVN-Kubernetes 集群网络供应商，则 kube-proxy 配置不会起作用。

defaultNetwork 对象配置
下表列出了 defaultNetwork 对象的值：

表 7.2. defaultNetwork 对象

字段	类型	描述
type	字符串	**OVNKubernetes**。Red Hat OpenShift Networking 网络插件在安装过程中被选择。此值在集群安装后无法更改。 **注意** OpenShift Container Platform 默认使用 OVN-Kubernetes 网络插件。OpenShift SDN 不再作为新集群的安装选择提供。
ovnKubernetesConfig	object	此对象仅对 OVN-Kubernetes 网络插件有效。

配置 OpenShift SDN 网络插件
下表描述了 OpenShift SDN 网络插件的配置字段：

表 7.3. openshiftSDNConfig object

字段	类型	描述
模式	字符串	OpenShift SDN 的网络隔离模式。
mtu	整数	VXLAN 覆盖网络的最大传输单元(MTU)。这个值通常是自动配置的。
vxlanPort	整数	用于所有 VXLAN 数据包的端口。默认值为 **4789**。

OpenShift SDN 配置示例

```
defaultNetwork:
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
```

配置 OVN-Kubernetes 网络插件
下表描述了 OVN-Kubernetes 网络插件的配置字段：

表 7.4. ovnKubernetesConfig object

字段	类型	描述
mtu	**整数**	Geneve（通用网络虚拟化封装）覆盖网络的最大传输单元 (MTU)。这个值通常是自动配置的。
genevePort	**整数**	Geneve 覆盖网络的 UDP 端口。
ipsecConfig	**object**	用于描述集群的 IPsec 模式的对象。
ipv4	**object**	为 IPv4 设置指定配置对象。
ipv6	**object**	为 IPv6 设置指定配置对象。
policyAuditConfig	**object**	指定用于自定义网络策略审计日志的配置对象。如果未设置，则使用默认的审计日志设置。
gatewayConfig	**object**	可选：指定一个配置对象来自定义如何将出口流量发送到节点网关。 **注意** 在迁移出口流量时，工作负载和服务流量会受到一定影响，直到 Cluster Network Operator (CNO) 成功推出更改。

表 7.5. **ovnKubernetesConfig.ipv4** object

字段	类型	描述
internalTransitSwitchSubnet	字符串	如果您的现有网络基础架构与 **100.88.0.0/16** IPv4 子网重叠，您可以指定不同的 IP 地址范围供 OVN-Kubernetes 使用。启用东西流量的分布式传输交换机的子网。此子网不能与 OVN-Kubernetes 或主机本身使用的任何其他子网重叠。必须足够大，以适应集群中的每个节点一个 IP 地址。 默认值为 **100.88.0.0/16**。
internalJoinSubnet	字符串	如果您的现有网络基础架构与 **100.64.0.0/16** IPv4 子网重叠，您可以指定不同的 IP 地址范围供 OVN-Kubernetes 使用。您必须确保 IP 地址范围没有与 OpenShift Container Platform 安装使用的任何其他子网重叠。IP 地址范围必须大于可添加到集群的最大节点数。例如，如果 **clusterNetwork.cidr** 值为 **10.128.0.0/14**，并且 **clusterNetwork.hostPrefix** 值为 **/23**，则最大节点数量为 **2^(23-14)=512**。 默认值为 **100.64.0.0/16**。

表 7.6. **ovnKubernetesConfig.ipv6** object

字段	类型	描述
internalTransitSwitchSubnet	字符串	如果您的现有网络基础架构与 **fd97::/64** IPv6 子网重叠，您可以指定不同的 IP 地址范围供 OVN-Kubernetes 使用。启用东西流量的分布式传输交换机的子网。此子网不能与 OVN-Kubernetes 或主机本身使用的任何其他子网重叠。必须足够大，以适应集群中的每个节点一个 IP 地址。 默认值为 **fd97::/64**。
internalJoinSubnet	字符串	如果您的现有网络基础架构与 **fd98::/64** IPv6 子网重叠，您可以指定不同的 IP 地址范围供 OVN-Kubernetes 使用。您必须确保 IP 地址范围没有与 OpenShift Container Platform 安装使用的任何其他子网重叠。IP 地址范围必须大于可添加到集群的最大节点数。 默认值为 **fd98::/64**。

表 7.7. **policyAuditConfig** object

字段	类型	描述
rateLimit	整数	每个节点每秒生成一次的消息数量上限。默认值为每秒 **20** 条消息。
maxFileSize	整数	审计日志的最大大小，以字节为单位。默认值为 **50000000** 或 50 MB。
maxLogFiles	整数	保留的日志文件的最大数量。
目的地	字符串	以下附加审计日志目标之一： **libc** 主机上的 journald 进程的 libc **syslog（）** 函数。 **UDP:<host>:<port>** 一个 syslog 服务器。将 **<host>:<port>** 替换为 **syslog 服务器的主机** 和端口。 **Unix:<file>** 由 **<file>** 指定的 Unix 域套接字文件。 **null** 不要将审计日志发送到任何其他目标。
syslogFacility	字符串	syslog 工具，如 as **kern**，如 RFC5424 定义。默认值为 **local0**。

表 7.8. **gatewayConfig** object

字段	类型	描述
routingViaHost	**布尔值**	将此字段设置为 **true**，将来自 pod 的出口流量发送到主机网络堆栈。对于依赖于在内核路由表中手动配置路由的高级别安装和应用程序，您可能需要将出口流量路由到主机网络堆栈。默认情况下，出口流量在 OVN 中进行处理以退出集群，不受内核路由表中的特殊路由的影响。默认值为 **false**。 此字段与 Open vSwitch 硬件卸载功能有交互。如果将此字段设置为 **true**，则不会获得卸载的性能优势，因为主机网络堆栈会处理出口流量。
ipForwarding	**object**	您可以使用 **Network** 资源中的 **ipForwarding** 规格来控制 OVN-Kubernetes 管理接口上所有流量的 IP 转发。指定 **Restricted** 只允许 Kubernetes 相关流量的 IP 转发。指定 **Global** 以允许转发所有 IP 流量。对于新安装，默认值为 **Restricted**。对于到 OpenShift Container Platform 4.14 或更高版本的更新，默认值为 **Global**。
ipv4	**object**	可选：指定一个对象来为主机配置内部 OVN-Kubernetes 伪装地址，以服务 IPv4 地址的流量。
ipv6	**object**	可选：指定一个对象来为主机配置内部 OVN-Kubernetes 伪装地址，以服务 IPv6 地址的流量。

表 7.9. gatewayConfig.ipv4 对象

字段	类型	描述
internalMasqueradeSubnet	**字符串**	内部使用的伪装 IPv4 地址，以启用主机服务流量。主机配置了这些 IP 地址和共享网关网桥接口。默认值为 **169.254.169.0/29**。

表 7.10. gatewayConfig.ipv6 对象

字段	类型	描述
internalMasqueradeSubnet	**字符串**	内部使用的伪装 IPv6 地址，以启用主机服务流量。主机配置了这些 IP 地址和共享网关网桥接口。默认值为 **fd69::/125**。

表 7.11. ipsecConfig 对象

字段	类型	描述

字段	类型	描述
模式	**字符串**	指定 IPsec 实现的行为。必须是以下值之一： • **Disabled**: 在集群节点上不启用 IPsec。 • **External** ：对于带有外部主机的网络流量，启用 IPsec。 • **Full**: IPsec 为带有外部主机的 pod 流量和网络流量启用 IPsec。

注意

您只能在集群安装过程中更改集群网络插件的配置，但 gatewayConfig 字段可作为安装后活动在运行时更改。

启用 IPSec 的 OVN-Kubernetes 配置示例

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
      ipsecConfig:
        mode: Full
```

重要

使用 OVNKubernetes 可能会导致 IBM Power® 上的堆栈耗尽问题。

kubeProxyConfig 对象配置（仅限 OpenShiftSDN 容器网络接口）

kubeProxyConfig 对象的值在下表中定义：

表 7.12. kubeProxyConfig object

字段	类型	描述
iptablesSyncPeriod	**字符串**	**iptables** 规则的刷新周期。默认值为 **30s**。有效的后缀包括 **s**、**m** 和 **h**，具体参见 Go 时间 包 文档。  **注意** 由于 OpenShift Container Platform 4.3 及更高版本中引进了性能改进，不再需要调整 **iptablesSyncPeriod** 参数。

字段	类型	描述
proxyArguments.iptables-min-sync-period	**array**	刷新 **iptables** 规则前的最短持续时间。此字段确保刷新的频率不会过于频繁。有效的后缀包括 **s**、**m** 和 **h**，具体参见 Go **time** 软件包。默认值为： kubeProxyConfig: proxyArguments: iptables-min-sync-period: - 0s

7.5.2. Cluster Network Operator 配置示例

以下示例中指定了完整的 CNO 配置：

Cluster Network Operator 对象示例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  networkType: OVNKubernetes
     clusterNetworkMTU: 8900
```

7.6. 其他资源

- operator.openshift.io API 组中的网络 API
- 修改 clusterNetwork IP 地址范围
- 从 OpenShift SDN 网络插件迁移

第 8 章 OPENSHIFT CONTAINER PLATFORM 中的 DNS OPERATOR

在 OpenShift Container Platform 中，DNS Operator 部署和管理 CoreDNS 实例，为集群中的 pod 提供名称解析服务，启用基于 DNS 的 Kubernetes 服务发现，并解析内部 cluster.local 名称。

8.1. 检查 DNS OPERATOR 的状态

DNS Operator 从 operator.openshift.io API 组实现 dns API。Operator 使用守护进程集部署 CoreDNS，为守护进程集创建一个服务，并将 kubelet 配置为指示 pod 使用 CoreDNS 服务 IP 地址进行名称解析。

流程

在安装过程中使用 Deployment 对象部署 DNS Operator。

1. 使用 oc get 命令查看部署状态：

```
$ oc get -n openshift-dns-operator deployment/dns-operator
```

输出示例

```
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
dns-operator  1/1     1            1           23h
```

2. 使用 oc get 命令来查看 DNS Operator 的状态：

```
$ oc get clusteroperator/dns
```

输出示例

```
NAME   VERSION      AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
dns    4.1.15-0.11  True        False         False      92m
```

AVAILABLE、PROGRESSING 和 DEGRADED 提供了有关 Operator 状态的信息。当 CoreDNS 守护进程中至少有一个 pod 被设置为 Available 状态时，AVAILABLE 为 True，DNS 服务有一个集群 IP 地址。

8.2. 查看默认 DNS

每个 OpenShift Container Platform 新安装都有一个名为 default 的 dns.operator。

流程

1. 使用 oc describe 命令来查看默认 dns：

```
$ oc describe dns.operator/default
```

输出示例

```
Name:         default
```

```
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         DNS
...
Status:
  Cluster Domain:  cluster.local (1)
  Cluster IP:      172.30.0.10 (2)
...
```

(1) Cluster Domain 字段是用来构造完全限定的 pod 和服务域名的基本 DNS 域。

(2) Cluster IP 是 Pod 为名称解析查询的地址。IP 由服务 CIDR 范围中的第 10 个地址定义。

2. 要查找集群的服务 CIDR 范围，请使用 oc get 命令：

```
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
```

输出示例

```
[172.30.0.0/16]
```

8.3. 使用 DNS 转发

您可以使用以下方法使用 DNS 转发来覆盖 /etc/resolv.conf 文件中的默认转发配置：

- 为每个区指定名称服务器 (spec.servers)。如果转发区是 OpenShift Container Platform 管理的 ingress 域，则必须为域授权上游名称服务器。
- 提供上游 DNS 服务器列表 (spec.upstreamResolvers)。
- 更改默认转发策略。

注意

默认域的 DNS 转发配置可以同时在 /etc/resolv.conf 文件和上游 DNS 服务器中指定默认服务器。

流程

1. 修改名为 default 的 DNS Operator 对象：

```
$ oc edit dns.operator/default
```

发出上一命令后，Operator 会根据 spec.servers 创建并更新名为 dns-default 的配置映射，并使用额外的服务器配置块。如果任何服务器都没有与查询匹配的区域，则名称解析会返回上游 DNS 服务器。

配置 DNS 转发

```
apiVersion: operator.openshift.io/v1
```

```
kind: DNS
metadata:
  name: default
spec:
  cache:
    negativeTTL: 0s
    positiveTTL: 0s
  logLevel: Normal
  nodePlacement: {}
  operatorLogLevel: Normal
  servers:
  - name: example-server 1
    zones:
    - example.com 2
    forwardPlugin:
      policy: Random 3
      upstreams: 4
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers: 5
    policy: Random 6
    protocolStrategy: "" 7
    transportConfig: {} 8
    upstreams:
    - type: SystemResolvConf 9
    - type: Network
      address: 1.2.3.4 10
      port: 53 11
    status:
      clusterDomain: cluster.local
      clusterIP: x.y.z.10
      conditions:
    ...
```

1 必须符合 rfc6335 服务名称语法。

2 必须符合 rfc1123 服务名称语法中的子域的定义。集群域 cluster.local 是对 zones 字段的无效子域。

3 定义用于选择 forwardPlugin 中列出的上游解析器的策略。默认值为 Random。您还可以使用 RoundRobin, 和 Sequential 值。

4 每个 forwardPlugin 最多允许 15 个upstreams。

5 您可以使用 upstreamResolvers 覆盖默认转发策略，并将 DNS 解析转发到默认域的指定 DNS 解析器（上游解析器）。如果没有提供任何上游解析器，DNS 名称查询将进入 /etc/resolv.conf 中声明的服务器。

6 决定选择上游中列出的 upstreams 服务器进行查询的顺序。您可以指定这些值之一：Random、RoundRobin 或 Sequential。默认值为 Sequential。

7 如果被省略，平台会选择一个默认值，通常是原始客户端请求的协议。设置为 TCP，以指定平台应该对所有上游 DNS 请求使用 TCP，即使客户端请求使用了 UDP。

8

用于配置传输类型、服务器名称和可选自定义 CA 或 CA 捆绑包，以便在将 DNS 请求转发到上游解析器时使用。

9 您可以指定两个类型的 upstreams：SystemResolvConf 或 Network。SystemResolvConf 将上游配置为使用 /etc/resolv.conf 和 Network 定义一个 Networkresolver。您可以指定其中一个或两者都指定。

10 如果指定类型是 Network，则必须提供 IP 地址。address 字段必须是有效的 IPv4 或 IPv6 地址。

11 如果指定类型是 Network，您可以选择性地提供端口。port 字段必须是 1 到 65535 之间的值。如果您没有为上游指定端口，则默认端口为 853。

其他资源

- 有关 DNS 转发的详情，请查看 CoreDNS 转发文档。

8.4. 检查 DNS OPERATOR 状态

您可以使用 oc describe 命令来检查状态并查看 DNS Operator 的详情。

流程

- 查看 DNS Operator 的状态：

```
$ oc describe clusteroperators/dns
```

虽然在特定版本中的信息和拼写可能有所不同，但预期的状态输出如下：

```
Status:
  Conditions:
    Last Transition Time:  <date>
    Message:               DNS "default" is available.
    Reason:                AsExpected
    Status:                True
    Type:                  Available
    Last Transition Time:  <date>
    Message:               Desired and current number of DNSes are equal
    Reason:                AsExpected
    Status:                False
    Type:                  Progressing
    Last Transition Time:  <date>
    Reason:                DNSNotDegraded
    Status:                False
    Type:                  Degraded
    Last Transition Time:  <date>
    Message:               DNS default is upgradeable: DNS Operator can be upgraded
    Reason:                DNSUpgradeable
    Status:                True
    Type:                  Upgradeable
```

8.5. 查看 DNS OPERATOR 日志

您可以使用 oc logs 命令来查看 DNS Operator 日志。

流程

- 查看 DNS Operator 的日志：

```
$ oc logs -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

8.6. 设置 COREDNS 日志级别

CoreDNS 和 CoreDNS Operator 的日志级别使用不同的方法设置。您可以配置 CoreDNS 日志级别来确定日志记录错误信息中的详情量。CoreDNS 日志级别的有效值为 Normal、Debug 和 Trace。默认 logLevel 为 Normal。

注意

CoreDNS 错误日志级别总是被启用。以下日志级别设置报告不同的错误响应：

- logLevel: Normal 启用 "errors" 类:log . { class error }.
- loglevel ：Debug 启用 "denial" 类:log . { class denial error }。
- logLevel: Trace 启用 "all" 类:log . { class all }.

流程

- 要将 logLevel 设置为 Debug，输入以下命令：

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --
type=merge
```

- 要将 logLevel 设置为 Trace，输入以下命令：

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --
type=merge
```

验证

- 要确保设置了所需的日志级别，请检查配置映射：

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

例如，在将 logLevel 设置为 Trace 后，您应该在每个 server 块中看到这个小节：

```
errors
log . {
    class all
}
```

8.7. 设置 COREDNS OPERATOR 的日志级别

CoreDNS 和 CoreDNS Operator 的日志级别使用不同的方法设置。集群管理员可以配置 Operator 日志

级别来更快地跟踪 OpenShift DNS 问题。operatorLogLevel 的有效值为 Normal、Debug 和 Trace。Trace 具有更详细的信息。默认 operatorlogLevel 为 Normal。Operator 问题有七个日志记录级别：Trace, Debug, Info, Warning, Error, Fatal, 和 Panic。设置了日志级别后，具有该严重性级别或以上级别的所有内容都会记录为日志条目。

- operatorLogLevel: "Normal" 设置 logrus.SetLogLevel("Info")。
- operatorLogLevel: "Debug" 设置 logrus.SetLogLevel("Debug")。
- operatorLogLevel: "Trace" 设置 logrus.SetLogLevel("Trace")。

流程

- 要将 operatorLogLevel 设置为 Debug，请输入以下命令：

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":
{"operatorLogLevel":"Debug"}}' --type=merge
```

- 要将 operatorLogLevel 设置为 Trace，请输入以下命令：

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":
{"operatorLogLevel":"Trace"}}' --type=merge
```

验证

1. 要查看生成的更改，请输入以下命令：

```
$ oc get dnses.operator -A -oyaml
```

您应该会看到两个日志级别条目。operatorLogLevel 适用于 OpenShift DNS Operator 问题，logLevel 适用于 CoreDNS pod 的 daemonset：

```
logLevel: Trace
operatorLogLevel: Debug
```

2. 要查看 daemonset 的日志，请输入以下命令：

```
$ oc logs -n openshift-dns ds/dns-default
```

8.8. 调整 COREDNS 缓存

对于 CoreDNS，您可以配置成功或不成功缓存的最长持续时间，也称为正缓存或负缓存。调整 DNS 查询响应的缓存持续时间可减少任何上游 DNS 解析器的负载。

警告

将 TTL 字段设为低值可能会导致集群、任何上游解析器或两者中负载的增加。

流程

1. 运行以下命令来编辑名为 **default** 的 DNS Operator 对象：

```
$ oc edit dns.operator.openshift.io/default
```

2. **修改生存时间 (TTL) 缓存值：**

配置 DNS 缓存

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h (1)
    negativeTTL: 0.5h10m (2)
```

(1) CoreDNS 会将字符串值 **1h** 转换为其相应的秒数。如果省略此字段，则假定该值为 **0s**，集群将使用内部默认值 **900s** 作为回退。

(2) 字符串值可以是单元的组合（如 **0.5h10m**），并被 CoreDNS 转换为相应秒数。如果省略了此字段，则假定该值为 **0s**，集群将使用内部默认值 **30s** 作为回退。

验证

1. 要查看更改，请运行以下命令再次查看配置映射：

```
oc get configmap/dns-default -n openshift-dns -o yaml
```

2. 验证您是否看到类似以下示例的条目：

```
    cache 3600 {
        denial 9984 2400
    }
```

其他资源

有关缓存的更多信息，请参阅 CoreDNS 缓存。

8.9. 高级任务

8.9.1. 更改 DNS Operator managementState

DNS Operator 管理 CoreDNS 组件，为集群中的 pod 和服务提供名称解析服务。默认情况下，DNS Operator 的 **managementState** 设置为 **Managed**，这意味着 DNS Operator 会主动管理其资源。您可以将其更改为 **Unmanaged**，这意味着 DNS Operator 不管理其资源。

以下是更改 DNS Operator **managementState** 的用例：

- 您是一个开发者，希望测试配置更改来查看它是否解决了 CoreDNS 中的问题。您可以通过将 **managementState** 设置为 **Unmanaged** 来停止 DNS Operator 覆盖配置更改。
- 您是一个集群管理员，报告了 CoreDNS 的问题，但在解决这个问题前需要应用一个临时解决方案。您可以将 DNS Operator 的 **managementState** 字段设置为 **Unmanaged** 以应用临时解决方案。

流程

1. 在 DNS Operator 中将 **managementState** 更改为 **Unmanaged** ：

```
oc patch dns.operator.openshift.io default --type merge --patch '{"spec":
{"managementState":"Unmanaged"}}'
```

2. 使用 **jsonpath** 命令行 JSON 解析器查看 DNS Operator 的**managementState** ：

```
$ oc get dns.operator.openshift.io default -ojsonpath='{.spec.managementState}'
```

输出示例

```
"Unmanaged"
```

注意

当 **managementState** 设置为 **Unmanaged** 时，您无法升级。

8.9.2. 控制 DNS pod 放置

DNS Operator 有两个守护进程集：一个用于 CoreDNS，名为 **dns-default**，另一个用于管理 名为 **node-resolver** 的 **/etc/hosts** 文件。

您可能会发现，需要控制哪些节点已分配并运行了 CoreDNS pod，尽管这不是一个常见操作。例如，如果集群管理员配置了可以禁止节点对节点间通信的安全策略，这需要限制 CoreDNS 运行 daemonset 的节点集合。如果 DNS pod 在集群中的某些节点上运行，且没有运行 DNS pod 的节点与运行 DNS pod 的节点具有网络连接，则 DNS 服务将适用于所有 pod。

node-resolver 守护进程集必须在每个节点主机上运行，因为它为集群镜像 registry 添加一个条目来支持拉取镜像。**node-resolver** pod 只有一个作业：查找 **image-registry.openshift-image-registry.svc** 服务的集群 IP 地址，并将其添加到节点主机上的 **/etc/hosts** 中，以便容器运行时可以解析服务名称。

作为集群管理员，您可以使用自定义节点选择器将 CoreDNS 的守护进程集配置为在某些节点上运行或不运行。

先决条件

- 已安装 **oc** CLI。
- 以具有 **cluster-admin** 权限的用户身份登录集群。
- 您的 DNS Operator **managementState** 设置为 **Managed**。

流程

- 要允许 CoreDNS 的守护进程集在特定节点上运行，请配置污点和容限：

1. 修改名为 default 的 DNS Operator 对象：

```
$ oc edit dns.operator/default
```

2. 为污点指定污点键和一个容忍度：

```
spec:
  nodePlacement:
    tolerations:
    - effect: NoExecute
      key: "dns-only"
      operators: Equal
      value: abc
      tolerationSeconds: 3600 1
```

1 如果污点是 dns-only，它可以无限期地被容许。您可以省略 tolerationSeconds。

8.9.3. 使用 TLS 配置 DNS 转发

在高度监管的环境中工作时，您可能需要在将请求转发到上游解析器时保护 DNS 流量，以便您可以确保额外的 DNS 流量和数据隐私。

请注意，CoreDNS 缓存在 10 秒内转发连接。如果没有请求，CoreDNS 将为该 10 秒打开 TCP 连接。对于大型集群，请确保您的 DNS 服务器知道可能有多个新的连接来保存打开，因为您可以在每个节点上启动连接。相应地设置 DNS 层次结构以避免性能问题。

流程

1. 修改名为 default 的 DNS Operator 对象：

```
$ oc edit dns.operator/default
```

集群管理员可以配置传输层安全(TLS)来转发 DNS 查询。

使用 TLS 配置 DNS 转发

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: example-server 1
    zones:
    - example.com 2
    forwardPlugin:
      transportConfig:
        transport: TLS 3
        tls:
          caBundle:
            name: mycacert
          serverName: dnstls.example.com 4
      policy: Random 5
```

```
    upstreams: 6
    - 1.1.1.1
    - 2.2.2.2:5353
  upstreamResolvers: 7
   transportConfig:
    transport: TLS
    tls:
      caBundle:
        name: mycacert
      serverName: dnstls.example.com
   upstreams:
   - type: Network 8
    address: 1.2.3.4 9
    port: 53 10
```

1. 必须符合 rfc6335 服务名称语法。
2. 必须符合 rfc1123 服务名称语法中的子域的定义。集群域 cluster.local 是对 zones 字段的无效子域。集群域 cluster.local 不是 zones 中的一个有效的 subdomain。
3. 在为转发 DNS 查询配置 TLS 时，将 transport 字段设置为具有值 TLS。
4. 当为转发 DNS 查询配置 TLS 时，这是用作服务器名称的一部分(SNI)的强制服务器名称来验证上游 TLS 服务器证书。
5. 定义用于选择上游解析器的策略。默认值为 Random。您还可以使用 RoundRobin, 和 Sequential 值。
6. 必需。使用它提供上游解析器。每个 forwardPlugin 条目最多允许 15 个upstreams 条目。
7. 可选。您可以使用它来覆盖默认策略，并将 DNS 解析转发到默认域的指定 DNS 解析器（上游解析器）。如果没有提供任何上游解析器，DNS 名称查询将进入 /etc/resolv.conf 中的服务器。
8. 在使用 TLS 时，只允许网络类型，且您必须提供 IP 地址。网络类型表示，该上游解析器应该独立于 /etc/resolv.conf 中列出的上游解析器单独处理转发请求。
9. address 字段必须是有效的 IPv4 或 IPv6 地址。
10. 您可以选择提供端口。port 必须是 1 到 65535 之间的值。如果您没有为上游指定端口，则默认端口为 853。

注意

如果 servers 未定义或无效，则配置映射只包括默认服务器。

验证

1. 查看配置映射：

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

基于 TLS 转发示例的 DNS ConfigMap 示例

```
apiVersion: v1
data:
  Corefile: |
    example.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
        forward . 3.3.3.3 4.4.4.4:5454 1
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf 1.2.3.4:53 {
            policy Random
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

1 对 forwardPlugin 的更改会触发 CoreDNS 守护进程集的滚动更新。

其他资源

- 有关 DNS 转发的详情，请查看 CoreDNS 转发文档。

第 9 章 OPENSHIFT CONTAINER PLATFORM 中的 INGRESS OPERATOR

9.1. OPENSHIFT CONTAINER PLATFORM INGRESS OPERATOR

在创建 OpenShift Container Platform 集群时，在集群中运行的 Pod 和服务会各自分配自己的 IP 地址。IP 地址可供附近运行的其他容器集和服务访问，但外部客户端无法访问这些 IP 地址。Ingress Operator 实现 **IngressController** API，是负责启用对 OpenShift Container Platform 集群服务的外部访问的组件。

Ingress Operator 通过部署和管理一个或多个基于 HAProxy 的 Ingress Controller 来处理路由，使外部客户端可以访问您的服务。您可以通过指定 OpenShift Container Platform **Route** 和 Kubernetes **Ingress** 资源，来使用 Ingress Operator 路由流量。Ingress Controller 中的配置（如定义 **endpointPublishingStrategy** 类型和内部负载平衡）提供了发布 Ingress Controller 端点的方法。

9.2. INGRESS 配置资产

安装程序在 **config.openshift.io** API 组中生成带有 **Ingress** 资源的资产，**cluster-ingress-02-config.yml**。

Ingress 资源的 YAML 定义

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshiftdemos.com
```

安装程序将这个资产保存在 **manifests/** 目录下的 **cluster-ingress-02-config.yml** 文件中。此 **Ingress** 资源定义 Ingress 的集群范围配置。此 Ingress 配置的用法如下所示：

- Ingress Operator 使用集群 Ingress 配置中的域，作为默认 Ingress Controller 的域。
- OpenShift API Server Operator 使用集群 Ingress 配置中的域。在为未指定显式主机的 **Route** 资源生成默认主机时，还会使用此域。

9.3. INGRESS CONTROLLER 配置参数

ingresscontrollers.operator.openshift.io 资源提供了以下配置参数。

参数	描述

参数	描述
domain	**domain** 是 Ingress Controller 服务的一个 DNS 名称，用于配置多个功能： ● 对于 **LoadBalancerService** 端点发布策略，**domain** 被用来配置 DNS 记录。请参阅 **endpointPublishingStrategy**。 ● 当使用生成的默认证书时，该证书对**域**及其**子域**有效。请参阅 **defaultCertificate**。 ● 该值会发布到独立的 Route 状态，以便用户了解目标外部 DNS 记录的位置。 **domain** 值在所有 Ingress 控制器中需要是唯一的，且不能更新。 如果为空，默认值为 **ingress.config.openshift.io/cluster .spec.domain**。
replicas	**replicas** 是 Ingress 控制器副本数量。如果没有设置，则默认值为 **2**。
endpointPublishingStrategy	**endpointPublishingStrategy** 用于向其他网络发布 Ingress Controller 端点，以启用负载均衡器集成，并提供对其他系统的访问。 在 GCP、AWS 和 Azure 上，您可以配置以下 **endpointPublishingStrategy** 字段： ● **loadBalancer.scope** ● **loadBalancer.allowedSourceRanges** 如果没有设置，则默认值基于 **infrastructure.config.openshift.io/cluster .status.platform**： ● Azure: **LoadBalancerService** （具有外部范围） ● Google Cloud Platform(GCP)： **LoadBalancerService** （具有外部范围） ● 裸机: **NodePortService** ● 其它: **HostNetwork**

参数	描述
	注意 **HostNetwork** 有一个 **hostNetwork** 字段，它有以下用于可选绑定端口的默认值：**httpPort: 80**,**httpsPort: 443**, 和 **statsPort: 1936**。使用绑定端口，您可以为 **HostNetwork** 策略在同一节点上部署多个 Ingress Controller。 **Example** `apiVersion: operator.openshift.io/v1` `kind: IngressController` `metadata:` `  name: internal` `  namespace: openshift-ingress-operator` `spec:` `  domain: example.com` `  endpointPublishingStrategy:` `  type: HostNetwork` `  hostNetwork:` `  httpPort: 80` `  httpsPort: 443` `  statsPort: 1936` **注意** 在 Red Hat OpenStack Platform (RHOSP) 上，只有云供应商配置为创建运行状况监视器时，才会支持 **LoadBalancerService** 端点发布策略。对于 RHOSP 16.2，只有在您使用 Amphora Octavia 供应商时，才能使用此策略。 如需更多信息，请参阅 RHOSP 安装文档中的"设置云供应商选项"部分。
defaultCertificate	**defaultCertificate** 的值是可以引用包含默认证书的 secret 的指代。当 Routes 没有指定自己的证书时，使用 **defaultCertificate**。 secret 必须包含以下密钥和数据：* **tls.crt**：证书文件内容 * **tls.key**：密钥文件内容 如果没有设置，则自动生成和使用通配符证书。该证书对 Ingress Controller 的**域** 和**子域**有效，所生成的证书的 CA 会自动与集群的信任存储集成。 内部证书（无论是生成的证书还是用户指定的证书）自动与 OpenShift Container Platform 内置的 OAuth 服务器集成。
namespaceSelector	**namespaceSelector** 用来过滤由 Ingress 控制器提供服务的一组命名空间。这对实现分片（shard）非常有用。
routeSelector	**routeSelector** 用于由 Ingress Controller 提供服务的一组 Routes。这对实现分片（shard）非常有用。

参数	描述
nodePlacement	**NodePlacement** 启用对 Ingress Controller 调度的显式控制。 如果没有设置，则使用默认值。  **注意** **nodePlacement** 参数包括两个部分: **nodeSelector** 和 **tolerations**。例如： nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists
tlsSecurityProfile	**tlsSecurityProfile** 指定 Ingress Controller 的 TLS 连接的设置。 如果没有设置，则默认值基于 **apiservers.config.openshift.io/cluster** 资源。 当使用 **Old**、**Intermediate** 和 **Modern** 配置集类型时，有效的配置集可能会在不同发行版本间有所改变。例如：使用在版本 **X.Y.Z** 中部署的 **Intermediate** 配置集，升级到版本 **X.Y.Z+1** 可能会导致新的配置集配置应用到 Ingress Controller，从而导致一个 rollout 操作。 Ingress Controller 的最低 TLS 版本是 **1.1**，最高 TLS 版本为 **1.3**。  **注意** 加密器和配置的安全配置集的最小 TLS 版本反映在 **TLSProfile** 状态中。  **重要** Ingress Operator 将 **Old** 或 **Custom** 配置集的 TLS **1.0** 转换为 **1.1**。

参数	描述
clientTLS	**clientTLS** 验证客户端对集群和服务的访问；因此，启用了 mutual TLS 身份验证。如果没有设置，则不启用客户端 TLS。 **clientTLS** 具有所需的子字段 **spec.clientTLS.clientCertificatePolicy** 和 **spec.clientTLS.ClientCA**。 **ClientCertificatePolicy** 子字段接受以下两个值之一：**Required** 或 **Optional**。**ClientCA** 子字段指定 openshift-config 命名空间中的配置映射。配置映射应包含 CA 证书捆绑包。 **AllowedSubjectPatterns** 是一个可选值，用于指定正则表达式列表，该列表与有效客户端证书上的可分辨名称匹配以过滤请求。正则表达式必须使用 PCRE 语法。至少一种模式必须与客户端证书的可分辨名称匹配；否则，入口控制器拒绝证书，并拒绝连接。如果没有指定，ingress 控制器不会根据可分辨的名称拒绝证书。
routeAdmission	**routeAdmission** 定义了处理新路由声明的策略，如允许或拒绝命名空间间的声明。 **namespaceOwnership** 描述了如何处理跨命名空间的主机名声明。默认为 **Strict**。 ● **Strict**：不允许路由在命名空间间声明相同的主机名。 ● **InterNamespaceAllowed** ：允许路由在命名空间间声明相同主机名的不同路径。 **wildcardPolicy** 描述了 Ingress Controller 如何处理采用通配符策略的路由。 ● **WildcardsAllowed**：表示 Ingress Controller 允许采用任何通配符策略的路由。 ● **WildcardsDisallowed**：表示 Ingress Controller 只接受采用 **None** 通配符策略的路由。将 **wildcardPolicy** 从 **WildcardsAllowed** 更新为 **WildcardsDisallowed**，会导致采用 **Subdomain** 通配符策略的已接受路由停止工作。这些路由必须重新创建为采用 **None** 通配符策略，让 Ingress Controller 重新接受。**WildcardsDisallowed** 是默认设置。

参数	描述
IngressControllerLogging	**logging** 定义了有关在哪里记录什么内容的参数。如果此字段为空，则会启用运行日志，但禁用访问日志。 • **access** 描述了客户端请求的日志记录方式。如果此字段为空，则禁用访问日志。 ◦ **destination** 描述日志消息的目的地。 ▪ **type** 是日志的目的地类型： • **Container** 指定日志应该进入 sidecar 容器。Ingress Operator 在 Ingress Controller pod 上配置名为 **logs** 的容器，并配置 Ingress Controller 以将日志写入容器。管理员应该配置一个自定义日志记录解决方案，从该容器读取日志。使用容器日志意味着，如果日志速率超过容器运行时或自定义日志解决方案的容量，则可能会出现日志丢失的问题。 • **Syslog** 指定日志发送到 Syslog 端点。管理员必须指定可以接收 Syslog 消息的端点。管理员应该已经配置了一个自定义 Syslog 实例。 ▪ **container** 描述了 **Container** 日志记录目的地类型的参数。目前没有容器日志记录参数，因此此字段必须为空。 ▪ **syslog** 描述了 **Syslog** 日志记录目的地类型的参数： • **address** 是接收日志消息的 syslog 端点的 IP 地址。 • **port** 是接收日志消息的 syslog 端点的 UDP 端口号。 • **MaxLength** 是 syslog 消息的最大长度。它必须介于 **480** 到 **4096** 字节之间。如果此字段为空，则最大长度设置为默认值 **1024** 字节。 • **facility** 指定日志消息的 syslog 工具。如果该字段为空，则工具为 **local1**。否则,它必须指定一个有效的 syslog 工具：**kern**、**user**、**mail**、**daemon**、**auth**、**syslog**, **lpr**, **news**, **uucp**, **cron**, **auth2**, **ftp**, **ntp**, **audit**, **alert**, **cron2**, **local0**, **local1**、**local2**、**local3**。**local4**、**local5**、**local6** 或 **local7**。 ◦ **httpLogFormat** 指定 HTTP 请求的日志消息格式。如果此字段为空，日志消息将使用实现中的默认 HTTP 日志格式。有关 HAProxy 的默认 HTTP 日志格式，请参阅 HAProxy 文档。

参数	描述
httpHeaders	**httpHeaders** 为 HTTP 标头定义策略。 通过为 **IngressControllerHTTPHeaders** 设置 **forwardHeaderPolicy**，您可以指定 Ingress 控制器何时和如何设置 **Forwarded**、**X-Forwarded-For**、**X-Forwarded-Host**、**X-Forwarded-Port**、**X-Forwarded-Proto** 和 **X-Forwarded-Proto-Version** HTTP 标头。 默认情况下，策略设置为 **Append**。 ● **Append** 指定 Ingress Controller 会附加标头，并保留任何现有的标头。 ● **Replace** 指定 Ingress Controller 设置标头，删除任何现有的标头。 ● **IfNone** 指定 Ingress Controller 在尚未设置标头时设置它们。 ● **Never** 指定 Ingress Controller 不会设置标头，并保留任何现有的标头。 通过设置 **headerNameCaseAdjustments**，您可以指定 HTTP 标头名对大小写的调整。每个调整都指定一个 HTTP 标头名称需要进行相关的大小写调整。例如，指定 **X-Forwarded-For** 表示 **x-forwarded-for** HTTP 标头应调整相应的大写。 这些调整仅应用于明文、边缘终止和重新加密路由，且仅在使用 HTTP/1 时有效。 对于请求标头，这些调整仅适用于具有 **haproxy.router.openshift.io/h1-adjust-case=true** 注解的路由。对于响应标头，这些调整适用于所有 HTTP 响应。如果此字段为空，则不会调整任何请求标头。 **actions** 指定对标头执行某些操作的选项。无法为 TLS 透传连接设置或删除标头。**actions** 字段具有额外的子字段 **spec.httpHeader.actions.response** 和 **spec.httpHeader.actions.request** ： ● **response** 子字段指定要设置或删除的 HTTP 响应标头列表。 ● **request** 子字段指定要设置或删除的 HTTP 请求标头列表。
httpCompression	**httpCompression** 定义 HTTP 流量压缩的策略。 ● **mimeTypes** 定义应该将压缩应用到的 MIME 类型列表。例如，**text/css; charset=utf-8**, **text/html**, **text/***, **image/svg+xml**, **application/octet-stream**, **X-custom/customsub**，格式为 **type/subtype; [;attribute=value]**。**types** 是：application, image, message, multipart, text, video, 或一个自定义类型（前面带有一个 **X-**；如需更详细的 MIME 类型和子类型的信息，请参阅 RFC1341
httpErrorCodePages	**httpErrorCodePages** 指定自定义 HTTP 错误代码响应页面。默认情况下，IngressController 使用 IngressController 镜像内构建的错误页面。

参数	描述
httpCaptureCookies	**httpCaptureCookies** 指定您要在访问日志中捕获的 HTTP cookie。如果 **httpCaptureCookies** 字段为空，则访问日志不会捕获 Cookie。 对于您要捕获的任何 Cookie，以下参数必须位于 **IngressController** 配置中： ● **name** 指定 Cookie 的名称。 ● **MaxLength** 指定 Cookie 的最大长度。 ● **matchType** 指定 Cookie 的 **name** 字段是否与捕获 Cookie 设置完全匹配，或者是捕获 Cookie 设置的前缀。**matchType** 字段使用 **Exact** 和 **Prefix** 参数。 例如： httpCaptureCookies: - matchType: Exact maxLength: 128 name: MYCOOKIE
httpCaptureHeaders	**httpCaptureHeaders** 指定要在访问日志中捕获的 HTTP 标头。如果 **httpCaptureHeaders** 字段为空，则访问日志不会捕获标头。 **httpCaptureHeaders** 包含两个要在访问日志中捕获的标头列表。这两个标题字段列表是 **request** 和 **response**。在这两个列表中，**name** 字段必须指定标头名称和 **maxlength** 字段，必须指定标头的最大长度。例如： httpCaptureHeaders: request: - maxLength: 256 name: Connection - maxLength: 128 name: User-Agent response: - maxLength: 256 name: Content-Type - maxLength: 256 name: Content-Length
tuningOptions	**tuningOptions** 指定用于调整 Ingress Controller pod 性能的选项。 ● **clientFinTimeout** 指定连接在等待客户端响应关闭连接时保持打开的时长。默认超时为 **1s**。 ● **clientTimeout** 指定连接在等待客户端响应时保持打开的时长。默认超时为 **30s**。 ● **headerBufferBytes** 为 Ingress Controller 连接会话指定保留多少内存（以字节为单位）。如果为 Ingress Controller 启用了 HTTP/2，则必须至少为 **16384**。如果没有设置，则默认值为 **32768** 字节。不建议设置此字段，因为 **headerBufferBytes** 值太小可能会破坏 Ingress Controller，而 **headerBufferBytes** 值过大可能会导致 Ingress Controller 使用比必要多的内存。

参数	描述
	• **headerBufferMaxRewriteBytes** 指定从 **headerBufferBytes** 为 Ingress Controller 连接会话保留多少内存（以字节为单位），用于 HTTP 标头重写和附加。**headerBufferMaxRewriteBytes** 的最小值是 **4096**。**headerBufferBytes** 必须大于 **headerBufferMaxRewriteBytes**，用于传入的 HTTP 请求。如果没有设置，则默认值为 **8192** 字节。不建议设置此字段，因为 **headerBufferMaxRewriteBytes** 值可能会破坏 Ingress Controller，**headerBufferMaxRewriteBytes** 值太大可能会导致 Ingress Controller 使用比必要大得多的内存。 • **healthCheckInterval** 指定路由器在健康检查之间等待的时间。默认值为 **5s**。 • **serverFinTimeout** 指定连接在等待服务器响应关闭连接时保持打开的时长。默认超时为 **1s**。 • **serverTimeout** 指定连接在等待服务器响应时保持打开的时长。默认超时为 **30s**。 • **threadCount** 指定每个 HAProxy 进程创建的线程数量。创建更多线程可让每个 Ingress Controller pod 处理更多连接，而代价会增加所使用的系统资源。HAProxy 支持多达 **64** 个线程。如果此字段为空，Ingress Controller 将使用默认值 **4** 个线程。默认值可能会在以后的版本中改变。不建议设置此字段，因为增加 HAProxy 线程数量可让 Ingress Controller pod 在负载下使用更多 CPU 时间，并阻止其他 pod 收到需要执行的 CPU 资源。减少线程数量可能会导致 Ingress Controller 执行不佳。 • **tlsInspectDelay** 指定路由器可以保存数据以查找匹配的路由的时长。如果把这个值设置得太短，对于 edge-terminated, reencrypted, 或 passthrough 的路由，则可能会导致路由器回退到使用默认证书，即使正在使用一个更加匹配的证书时也是如此。默认检查延迟为 **5s**。 • **tunnelTimeout** 指定隧道连接在隧道闲置期间保持打开的时长，包括 websockets。默认超时为 **1h**。 • **maxConnections** 指定每个 HAProxy 进程可建立的最大同时连接数。增加这个值可让每个入口控制器 pod 以额外的系统资源成本处理更多连接。允许的值是 **0**、**-1**、以及范围为 **2000** 和 **2000000** 内的任何值，或者字段可以留空。 ◦ 如果此字段留空，或者值为 **0**，Ingress Controller 将使用默认值 **50000**。这个值可能在以后的版本中有所改变。 ◦ 如果字段的值为 **-1**，则 HAProxy 将根据运行中容器中的可用 **ulimits** 动态计算最大值。与当前默认值 **50000** 相比，此进程会产生很大的内存用量。 ◦ 如果字段的值大于当前操作系统的限制，则 HAProxy 进程将不会启动。 ◦ 如果您选择了一个离散值，并且路由器 pod 迁移到新节点，则新节点可能没有配置相同的 **ulimit**。在这种情况下，pod 无法启动。 ◦ 如果您配置了不同的 **ulimits** 的节点，并且您选择离散值，则建议为该字段使用 **-1** 的值，以便在运行时计算连接的最大数量。

参数	描述
logEmptyRequests	**logEmptyRequests** 指定没有接收和记录请求的连接。这些空请求来自负载均衡器健康探测或 Web 浏览器规范连接(preconnect)，并记录这些请求。但是，这些请求可能是由网络错误导致的，在这种情况下，记录空请求可用于诊断错误。这些请求可能是由端口扫描导致的，记录空请求有助于检测入侵尝试。此字段允许的值有 **Log** 和 **Ignore**。默认值为 **Log**。 **LoggingPolicy** 类型接受以下两个值之一： • **log** ：将此值设置为 **Log** 表示应记录某一事件。 • **ignore** ：将此值设置为 **Ignore** 会在 HAproxy 配置中设置 **dontlognull** 选项。
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy** 描述了在收到请求前发生超时时，如何处理 HTTP 连接。此字段允许的值是 **Respond** 和 **Ignore**。默认值为 **Respond**。 **HTTPEmptyRequestsPolicy** 类型接受以下两个值之一： • **Respond**：如果字段设置为 **Respond**，Ingress Controller 会发送 HTTP **400** 或 **408** 响应，在启用了访问日志时记录连接，并在适当的指标中计数连接。 • **ignore**：将这个选项设置为 **Ignore** 会在 HAproxy 配置中添加 **http-ignore-probes** 参数。如果字段设置为 **Ignore**，Ingnore 会在不发送响应的情况下关闭连接，然后记录连接或递增指标。 这些连接来自负载均衡器健康探测或 Web 浏览器规范连接（预连接），可以安全地忽略。但是，这些请求可能是由网络错误造成的，因此将此字段设置为 **Ignore** 可能会妨碍对问题的检测和诊断。这些请求可能是由端口扫描导致的，在这种情况下，记录空请求有助于检测入侵尝试。

注意

所有参数都是可选的。

9.3.1. Ingress Controller TLS 安全配置集

TLS 安全配置文件为服务器提供了一种方式，以规范连接的客户端在连接服务器时可以使用哪些密码。

9.3.1.1. 了解 TLS 安全配置集

您可以使用 TLS（Transport Layer Security）安全配置集来定义各种 OpenShift Container Platform 组件需要哪些 TLS 密码。OpenShift Container Platform TLS 安全配置集基于 Mozilla 推荐的配置。

您可以为每个组件指定以下 TLS 安全配置集之一：

表 9.1. TLS 安全配置集

profile	描述

profile	描述
Old	此配置集用于旧的客户端或库。该配置集基于旧的向后兼容性建议配置。 **Old** 配置集要求最低 TLS 版本 1.0。 **注意** 对于 Ingress Controller，最小 TLS 版本从 1.0 转换为 1.1。
Intermediate	这个配置集是大多数客户端的建议配置。它是 Ingress Controller、kubelet 和 control plane 的默认 TLS 安全配置集。该配置集基于 Intermediate 兼容性推荐的配置。 **Intermediate** 配置集需要最小 TLS 版本 1.2。
Modern	此配置集主要用于不需要向后兼容的现代客户端。这个配置集基于 Modern 兼容性推荐的配置。 **Modern** 配置集需要最低 TLS 版本 1.3。
Custom	此配置集允许您定义要使用的 TLS 版本和密码。 **警告** 使用 **Custom** 配置集时要谨慎，因为无效的配置可能会导致问题。

注意

当使用预定义的配置集类型时，有效的配置集配置可能会在发行版本之间有所改变。例如，使用在版本 X.Y.Z 中部署的 Intermediate 配置集指定了一个规格，升级到版本 X.Y.Z+1 可能会导致应用新的配置集配置，从而导致推出部署。

9.3.1.2. 为 Ingress Controller 配置 TLS 安全配置集

要为 Ingress Controller 配置 TLS 安全配置集，请编辑 IngressController 自定义资源（CR）来指定预定义或自定义 TLS 安全配置集。如果没有配置 TLS 安全配置集，则默认值基于为 API 服务器设置的 TLS 安全配置集。

配置 Old TLS 安全配置集的 IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
```

```
    old: {}
    type: Old
...
```

TLS 安全配置集定义 Ingress Controller 的 TLS 连接的最低 TLS 版本和 TLS 密码。

您可以在 Status.Tls Profile 和 Spec.Tls Security Profile 下看到 IngressController 自定义资源（CR）中配置的 TLS 安全配置集的密码和最小 TLS 版本。对于 Custom TLS 安全配置集，这两个参数下列出了特定的密码和最低 TLS 版本。

注意

HAProxy Ingress Controller 镜像支持 TLS 1.3 和 Modern 配置集。

Ingress Operator 还会将 Old 或 Custom 配置集的 TLS 1.0 转换为 1.1。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 编辑 openshift-ingress-operator 项目中的 IngressController CR，以配置 TLS 安全配置集：

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. 添加 spec.tlsSecurityProfile 字段：

   Custom 配置集的 IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
...
```

(1) 指定 TLS 安全配置集类型（Old、Intermediate 或 Custom）。默认值为 Intermediate。

(2) 为所选类型指定适当的字段：

- old: {}
- intermediate: {}
- custom:

3 对于 custom 类型，请指定 TLS 密码列表和最低接受的 TLS 版本。

3. 保存文件以使改变生效。

验证

- 验证 IngressController CR 中是否设置了配置集：

```
$ oc describe IngressController default -n openshift-ingress-operator
```

输出示例

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

9.3.1.3. 配置 mutual TLS 身份验证

您可以通过设置 spec.clientTLS 值，将 Ingress Controller 配置为启用 mutual TLS (mTLS) 身份验证。clientTLS 值将 Ingress Controller 配置为验证客户端证书。此配置包括设置clientCA 值，这是对配置映射的引用。配置映射包含 PEM 编码的 CA 证书捆绑包，用于验证客户端的证书。另外，您还可以配置证书主题过滤器列表。

如果 clientCA 值指定了 X509v3 证书撤销列表 (CRL) 分发点，Ingress Operator 会下载并管理基于每个提供的证书中指定的 HTTP URI X509v3 CRL 分发点的 CRL 配置映射。Ingress Controller 在 mTLS/TLS 协商过程中使用此配置映射。不提供有效证书的请求将被拒绝。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 您有一个 PEM 编码的 CA 证书捆绑包。
- 如果您的 CA 捆绑包引用 CRL 发布点，还必须将最终用户或叶证书包含在客户端 CA 捆绑包中。此证书必须在 CRL 分发点下包含 HTTP URI，如 RFC 5280 所述。例如：

```
Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
```

```
        Subject: SOME SIGNED CERT        X509v3 CRL Distribution Points:
            Full Name:
              URI:http://crl.example.com/example.crl
```

流程

1. 在 openshift-config 命名空间中，从 CA 捆绑包创建配置映射：

```
$ oc create configmap \
  router-ca-certs-default \
  --from-file=ca-bundle.pem=client-ca.crt \ 1
  -n openshift-config
```

1 配置映射数据键必须是 ca-bundle.pem，数据值必须是 PEM 格式的 CA 证书。

2. 编辑 openshift-ingress-operator 项目中的 IngressController 资源：

```
$ oc edit IngressController default -n openshift-ingress-operator
```

3. 添加 spec.clientTLS 字段和子字段来配置 mutual TLS：

   指定过滤模式的 clientTLS 配置集的 IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
    allowedSubjectPatterns:
    - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

4. 可选，输入以下命令获取 allowedSubjectPatterns 的可辨识名称 (DN)。

```
$ openssl  x509 -in custom-cert.pem  -noout -subject
subject= /CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift
```

9.4. 查看默认的 INGRESS CONTROLLER

Ingress Operator 是 OpenShift Container Platform 的一个核心功能，开箱即用。

每个 OpenShift Container Platform 新安装都有一个名为 default 的 ingresscontroller。它可以通过额外的 Ingress Controller 来补充。如果删除了默认的 ingresscontroller，Ingress Operator 会在一分钟内自动重新创建。

流程

- 查看默认的 Ingress Controller：

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
```

9.5. 查看 INGRESS OPERATOR 状态

您可以查看并检查 Ingress Operator 的状态。

流程

- 查看您的 Ingress Operator 状态：

```
$ oc describe clusteroperators/ingress
```

9.6. 查看 INGRESS CONTROLLER 日志

您可以查看 Ingress Controller 日志。

流程

- 查看 Ingress Controller 日志：

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c
<container_name>
```

9.7. 查看 INGRESS CONTROLLER 状态

您可以查看特定 Ingress Controller 的状态。

流程

- 查看 Ingress Controller 的状态：

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
```

9.8. 创建自定义 INGRESS CONTROLLER

作为集群管理员，您可以创建新的自定义 Ingress Controller。因为默认 Ingress Controller 在 OpenShift Container Platform 更新过程中可能会改变，所以创建自定义 Ingress Controller 以在集群更新中手动保留配置会很有用。

这个示例为自定义 Ingress Controller 提供最小规格。要进一步定制您的自定义 Ingress Controller，请参阅"配置 Ingress Controller"。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建定义自定义 IngressController 对象的 YAML 文件：

custom-ingress-controller.yaml 文件示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <custom_name> 1
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: <custom-ingress-custom-certs> 2
  replicas: 1 3
  domain: <custom_domain> 4
```

1 为 IngressController 对象指定自定义名称。

2 使用自定义通配符证书指定 secret 名称。

3 最小副本需要是 ONE

4 指定您的域名的域。IngressController 对象中指定的域以及用于证书的域必须匹配。例如，如果 domain 值为 "custom_domain.mycompany.com"，则证书必须具有 SAN *.custom_domain.mycompany.com（在域中添加了 *.）。

2. 运行以下命令来创建对象：

```
$ oc create -f custom-ingress-controller.yaml
```

9.9. 配置 INGRESS CONTROLLER

9.9.1. 设置自定义默认证书

作为管理员，您可以通过创建 Secret 资源并编辑 IngressController 自定义资源 (CR)，将 Ingress Controller 配置为使用自定义证书。

先决条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中该证书由可信证书认证机构签名，或者由您在一个自定义 PKI 中配置的私有可信证书认证机构签名。
- 您的证书满足以下要求：
  - 该证书对入口域有效。
  - 证书使用 subjectAltName 扩展来指定通配符域，如 *.apps.ocp4.example.com。
- 您必须有一个 IngressController CR。您可以使用默认值：

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

输出示例

```
NAME      AGE
default   10m
```

注意

如果您有中间证书，则必须将其包含在包含自定义默认证书的 secret 的 tls.crt 文件中。指定证书时指定的顺序是相关的; 在任意服务器证书后列出您的中间证书。

流程

以下步骤假定自定义证书和密钥对位于当前工作目录下的 tls.crt 和 tls.key 文件中。替换 tls.crt 和 tls.key 的实际路径名。在创建 Secret 资源并在 IngressController CR 中引用它时，您也可以将 custom-certs-default 替换成另一名称。

注意

此操作会导致使用滚动部署策略重新部署 Ingress Controller。

1. 使用 tls.crt 和 tls.key 文件，创建在 openshift-ingress 命名空间中包含自定义证书的 Secret 资源。

```
$ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt
--key=tls.key
```

2. 更新 IngressController CR，以引用新的证书 Secret：

```
$ oc patch --type=merge --namespace openshift-ingress-operator
ingresscontrollers/default \
  --patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
```

3. 验证更新是否已生效：

```
$ echo Q |\
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null |\
  openssl x509 -noout -subject -issuer -enddate
```

其中：

<domain>
指定集群的基域名。

输出示例

```
subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
notAfter=May 10 08:32:45 2022 GM
```

提示

您还可以应用以下 YAML 来设置自定义默认证书：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: custom-certs-default
```

证书 Secret 名称应该与用来更新 CR 的值匹配。

修改了 IngressController CR 后，Ingress Operator 将更新 Ingress Controller 的部署以使用自定义证书。

9.9.2. 删除自定义默认证书

作为管理员，您可以删除配置了 Ingress Controller 的自定义证书。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。
- 您之前为 Ingress Controller 配置了自定义默认证书。

流程

- 要删除自定义证书并恢复 OpenShift Container Platform 附带的证书，请输入以下命令：

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
  --type json -p $'- op: remove\n  path: /spec/defaultCertificate'
```

集群协调新证书配置时可能会有延迟。

验证

- 要确认原始集群证书已被恢复，请输入以下命令：

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

其中：

<domain>
  指定集群的基域名。

输出示例

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

9.9.3. 自动扩展 Ingress Controller

自动缩放 Ingress Controller 以动态满足路由性能或可用性要求，如提高吞吐量的要求。以下流程提供了扩展默认 IngressController 的示例。

先决条件

1. 已安装 OpenShift CLI (oc)。
2. 您可以使用具有 cluster-admin 角色的用户访问 OpenShift Container Platform 集群。
3. 已安装自定义 Metrics Autoscaler Operator。
4. 您位于 openshift-ingress-operator 项目命名空间中。

流程

1. 运行以下命令，创建一个服务帐户来与 Thanos 进行身份验证：

```
$ oc create serviceaccount thanos && oc describe serviceaccount thanos
```

输出示例

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-b4l9s
Mountable secrets:   thanos-dockercfg-b4l9s
Tokens:              thanos-token-c422q
Events:              <none>
```

2. 使用服务帐户的令牌，在 openshift-ingress-operator 命名空间中定义一个 TriggerAuthentication 对象。
   a. 运行以下命令，定义包含 secret 的变量 secret：

```
$ secret=$(oc get secret | grep thanos-token | head -n 1 | awk '{ print $1 }')
```

   b. 创建 TriggerAuthentication 对象，并将 secret 变量的值传递给 TOKEN 参数：

```
$ oc process TOKEN="$secret" -f - <<EOF | oc apply -f -
apiVersion: template.openshift.io/v1
kind: Template
parameters:
- name: TOKEN
objects:
- apiVersion: keda.sh/v1alpha1
  kind: TriggerAuthentication
```

```
  metadata:
    name: keda-trigger-auth-prometheus
  spec:
    secretTargetRef:
    - parameter: bearerToken
      name: \${TOKEN}
      key: token
    - parameter: ca
      name: \${TOKEN}
      key: ca.crt
EOF
```

3. 创建并应用角色以从 Thanos 读取指标：

   a. 创建一个新角色 thanos-metrics-reader.yaml，从 pod 和节点读取指标：

   thanos-metrics-reader.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
```

   b. 运行以下命令来应用新角色：

```
$ oc apply -f thanos-metrics-reader.yaml
```

4. 输入以下命令在服务帐户中添加新角色：

```
$ oc adm policy add-role-to-user thanos-metrics-reader -z thanos --role-
namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-
monitoring-view -z thanos
```

注意

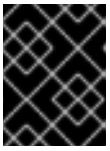
只有在使用跨命名空间查询时，才需要参数 add-cluster-role-to-user。以下步骤使用 kube-metrics 命名空间中的查询，该命名空间需要此参数。

5. 创建一个新的 ScaledObject YAML 文件 ingress-autoscaler.yaml，该文件以默认 Ingress Controller 部署为目标：

ScaledObject 定义示例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
spec:
  scaleTargetRef: 1
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 2
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091
3
      namespace: openshift-ingress-operator 4
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' 5
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

1. 要目标的自定义资源。在本例中，Ingress Controller。
2. 可选：最大副本数。如果省略此字段，则默认最大值设置为 100 个副本。
3. openshift-monitoring 命名空间中的 Thanos 服务端点。
4. Ingress Operator 命名空间。
5. 此表达式评估为，部署的集群中存在很多 worker 节点。



重要

如果使用跨命名空间查询，您必须在 serverAddress 字段中目标端口 9091 而不是端口 9092。您还必须有升级的特权，才能从此端口读取指标。

6. **运行以下命令来应用自定义资源定义：**

```
$ oc apply -f ingress-autoscaler.yaml
```

验证

- **运行以下命令，验证默认 Ingress Controller 是否已扩展以匹配 kube-state-metrics 查询返回的值：**
  - **使用 grep 命令搜索 Ingress Controller YAML 文件以查找副本：**

```
$ oc get ingresscontroller/default -o yaml | grep replicas:
```

输出示例

```
replicas: 3
```

  - **获取 openshift-ingress 项目中的 pod：**

```
$ oc get pods -n openshift-ingress
```

输出示例

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
router-default-7b5df44ff-wwsth   2/2     Running   0          66s
```

其他资源

- 为用户定义的项目启用监控
- 安装自定义指标自动扩展
- 了解自定义指标自动扩展触发器身份验证
- 配置自定义指标自动扩展以使用 OpenShift Container Platform 监控
- 了解如何添加自定义指标自动扩展

9.9.4. 扩展 Ingress Controller

手动扩展 Ingress Controller 以满足路由性能或可用性要求，如提高吞吐量的要求。oc 命令用于扩展 IngressController 资源。以下流程提供了扩展默认 IngressController 的示例。

注意

扩展不是立刻就可以完成的操作，因为它需要时间来创建所需的副本数。

流程

1. 查看默认 IngressController 的当前可用副本数：

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

输出示例

```
2
```

2. 使用 oc patch 命令，将默认 IngressController 扩展至所需的副本数。以下示例将默认 IngressController 扩展至 3 个副本：

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":
{"replicas": 3}}' --type=merge
```

输出示例

```
ingresscontroller.operator.openshift.io/default patched
```

3. 验证默认 IngressController 是否已扩展至您指定的副本数：

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

输出示例

```
3
```

提示

您还可以应用以下 YAML 将 Ingress Controller 扩展为三个副本：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3                    1
```

1 如果需要不同数量的副本，请更改 replicas 值。

9.9.5. 配置 Ingress 访问日志

您可以配置 Ingress Controller 以启用访问日志。如果您的集群没有接收许多流量，那么您可以将日志记录到 sidecar。如果您的集群接收大量流量，为了避免超出日志记录堆栈的容量，或与 OpenShift Container Platform 之外的日志记录基础架构集成，您可以将日志转发到自定义 syslog 端点。您还可以指定访问日志的格式。

当不存在 Syslog 日志记录基础架构时，容器日志记录可用于在低流量集群中启用访问日志，或者在诊断 Ingress Controller 时进行简短使用。

对于访问日志可能会超过 OpenShift Logging 堆栈容量的高流量集群，或需要任何日志记录解决方案与现有 Syslog 日志记录基础架构集成的环境，则需要 syslog。Syslog 用例可能会相互重叠。

先决条件

- 以具有 cluster-admin 特权的用户身份登录。

流程

配置 Ingress 访问日志到 sidecar。

- 要配置 Ingress 访问日志记录，您必须使用 spec.logging.access.destination 指定一个目的地。要将日志记录指定到 sidecar 容器，您必须指定 Container spec.logging.access.destination.type。以下示例是将日志记录到 Container 目的地的 Ingress Controller 定义：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
```

- 当将 Ingress Controller 配置为日志记录到 sidecar 时，Operator 会在 Ingress Controller Pod 中创建一个名为 logs 的容器：

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

输出示例

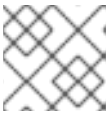
```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080 0/0/1/0/1 200 142 - - --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

配置 Ingress 访问日志记录到 Syslog 端点。

- 要配置 Ingress 访问日志记录，您必须使用 spec.logging.access.destination 指定一个目的地。要将日志记录指定到 Syslog 端点目的地，您必须为 spec.logging.access.destination.type 指定 Syslog。如果目的地类型是 Syslog，则必须使用

spec.logging.access.destination.syslog.endpoint 指定一个目的地端点，并可使用 spec.logging.access.destination.syslog.facility 指定一个工具。以下示例是将日志记录到 Syslog 目的地的 Ingress Controller 定义：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```



注意

Syslog 目的地端口必须是 UDP。

使用特定的日志格式配置 Ingress 访问日志。

- 您可以指定 spec.logging.access.httpLogFormat 来自定义日志格式。以下示例是一个 Ingress Controller 定义，它将日志记录到 IP 地址为 1.2.3.4、端口为 10514 的 syslog 端点：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
      httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

禁用 Ingress 访问日志。

- 要禁用 Ingress 访问日志，请保留 spec.logging 或 spec.logging.access 为空：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
```

```
  replicas: 2
  logging:
    access: null
```

允许 Ingress Controller 在使用 sidecar 时，修改 HAProxy 日志长度。

- 如果您使用 spec.logging.access.destination.syslog.maxLength，请使用 spec.logging.access.destination.type: Syslog。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514
```

- 如果您使用 spec.logging.access.destination.container.maxLength，请使用 spec.logging.access.destination.type: Container。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
        container:
          maxLength: 8192
```

9.9.6. 设置 Ingress Controller 线程数

集群管理员可设置线程数，以增加集群可以处理的入站的连接量。您可以修补现有的 Ingress Controller 来增加线程量。

先决条件

- 以下假设您已创建了 Ingress Controller。

流程

- 更新 Ingress Controller 以增加线程数量：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p
'{"spec":{"tuningOptions": {"threadCount": 8}}}'
```

注意

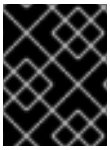
如果您的节点有能力运行大量资源，您可以使用与预期节点容量匹配的标签配置 spec.nodePlacement.nodeSelector，并将 spec.tuningOptions.threadCount 配置为一个适当的高值。

9.9.7. 配置 Ingress Controller 以使用内部负载均衡器

当在云平台上创建 Ingress Controller 时，Ingress Controller 默认由一个公共云负载均衡器发布。作为管理员，您可以创建一个使用内部云负载均衡器的 Ingress Controller。

警告

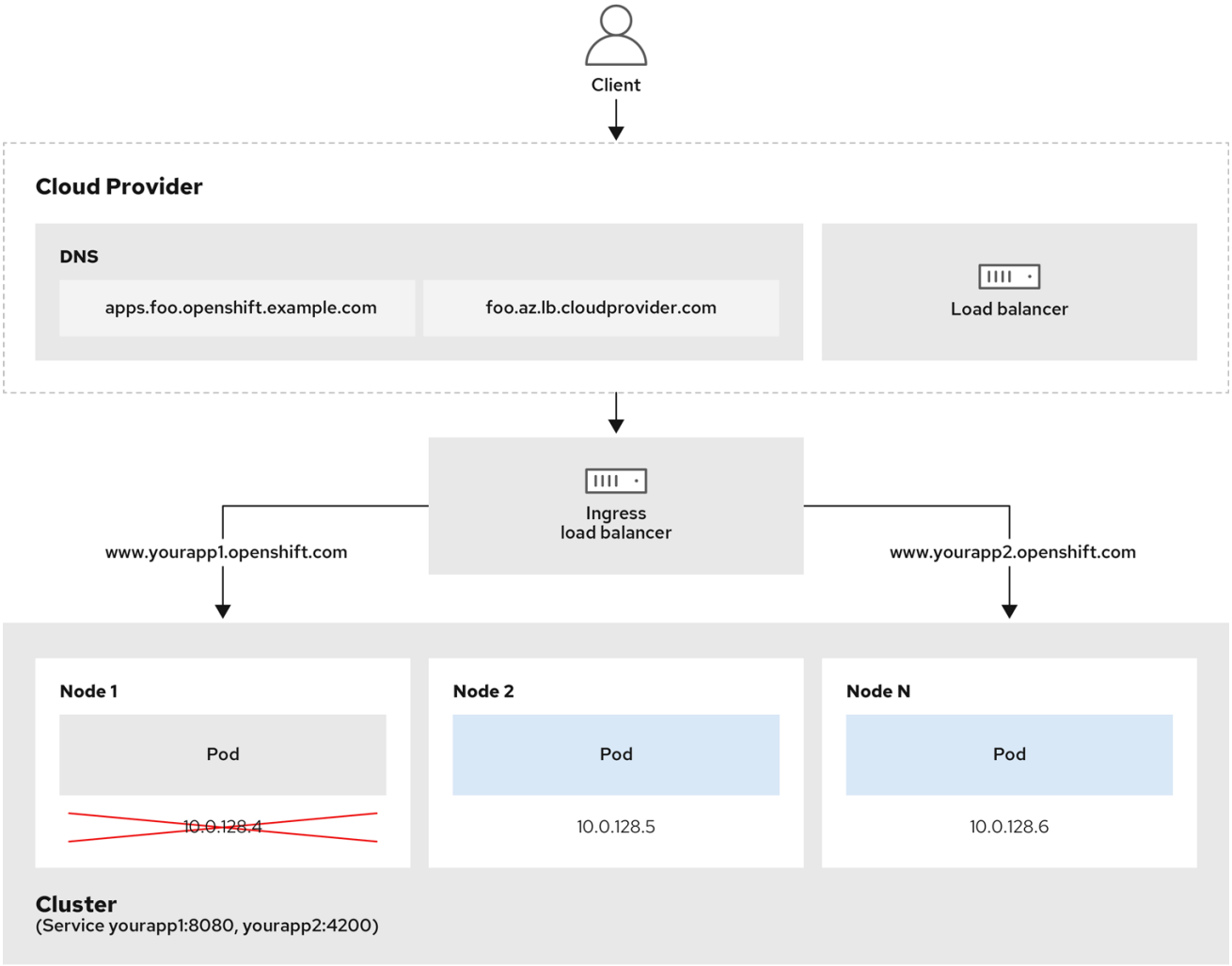
如果云供应商是 Microsoft Azure，则必须至少有一个指向节点的公共负载均衡器。如果不这样做，所有节点都将丢失到互联网的出站连接。



重要

如果要更改 IngressController 的 scope，您可以在创建自定义资源(CR)后更改 .spec.endpointPublishingStrategy.loadBalancer.scope 参数。

图 9.1. LoadBalancer 图表



上图显示了与 OpenShift Container Platform Ingress LoadBalancerService 端点发布策略相关的以下概念：

- 您可以使用 OpenShift Ingress Controller Load Balancer 在外部使用云供应商负载均衡器或内部加载负载。
- 您可以使用负载均衡器的单个 IP 地址以及更熟悉的端口，如 8080 和 4200，如图形中所述的集群所示。
- 来自外部负载均衡器的流量定向到 pod，并由负载均衡器管理，如下节点的实例中所述。有关实现详情请查看 Kubernetes 服务文档。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在名为 <name>-ingress-controller.yaml 的文件中创建 IngressController 自定义资源 (CR)，如下例所示：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal 3
```

1. 将 <name> 替换为 IngressController 对象的名称。
2. 指定控制器发布的应用程序的 domain。
3. 指定一个 Internal 值以使用内部负载均衡器。

2. 运行以下命令，创建上一步中定义的 Ingress Controller：

```
$ oc create -f <name>-ingress-controller.yaml 1
```

1. 将 <name> 替换为 IngressController 对象的名称。

3. 可选：通过运行以下命令确认创建了 Ingress Controller：

```
$ oc --all-namespaces=true get ingresscontrollers
```

9.9.8. 在 GCP 上为 Ingress Controller 配置全局访问

在带有一个内部负载均衡器的 GCP 上创建的 Ingress Controller 会为服务生成一个内部 IP 地址。集群管理员可指定全局访问选项，该选项可启用同一 VPC 网络内任何区域中的客户端作为负载均衡器，以访问集群上运行的工作负载。

如需更多信息，请参阅 GCP 文档以了解全局访问。

先决条件

- 您已在 GCP 基础架构上部署了 OpenShift Container Platform 集群。
- 已将 Ingress Controller 配置为使用内部负载均衡器。
- 已安装 OpenShift CLI（oc）。

流程

1. 配置 Ingress Controller 资源，以允许全局访问。

注意

您还可以创建 Ingress Controller 并指定全局访问选项。

a. 配置 Ingress Controller 资源：

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

b. 编辑 YAML 文件：

clientAccess 配置为 Global 的示例

```
spec:
  endpointPublishingStrategy:
    loadBalancer:
      providerParameters:
        gcp:
          clientAccess: Global 1
        type: GCP
      scope: Internal
    type: LoadBalancerService
```

1 将 gcp.clientAccess 设置为 Global。

c. 保存文件以使改变生效。

2. 运行以下命令，以验证该服务是否允许全局访问：

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

输出显示，全局访问已为带有注解 networking.gke.io/internal-load-balancer-allow-global-access 的 GCP 启用。

9.9.9. 设置 Ingress Controller 健康检查间隔

集群管理员可以设置健康检查间隔，以定义路由器在两个连续健康检查之间等待的时间。这个值会作为所有路由的默认值进行全局应用。默认值为 5 秒。

先决条件

- 以下假设您已创建了 Ingress Controller。

流程

- 更新 Ingress Controller，以更改后端健康检查之间的间隔：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p
'{"spec":{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```

注意

要覆盖单个路由的 healthCheckInterval，请使用路由注解 router.openshift.io/haproxy.health.check.interval

9.9.10. 将集群的默认 Ingress Controller 配置为内部

您可以通过删除并重新它来将默认 Ingress Controller 配置为内部。

警告

如果云供应商是 Microsoft Azure，则必须至少有一个指向节点的公共负载均衡器。如果不这样做，所有节点都将丢失到互联网的出站连接。

重要

如果要更改 IngressController 的 scope，您可以在创建自定义资源(CR)后更改 .spec.endpointPublishingStrategy.loadBalancer.scope 参数。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 通过删除并重新创建集群，将 默认 Ingress Controller 配置为内部。

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

9.9.11. 配置路由准入策略

管理员和应用程序开发人员可在多个命名空间中运行具有相同域名的应用程序。这是针对多个团队开发的、在同一个主机名上公开的微服务的机构。

警告

只有在命名空间间有信任的集群才会启用跨命名空间之间的声明，否则恶意用户可能会接管主机名。因此，默认的准入策略不允许在命名空间间声明主机名。

先决条件

- **必须具有集群管理员权限。**

流程

- **使用以下命令编辑 ingresscontroller 资源变量的.spec.** routeAdmission **字段：**

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
{"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --
type=merge
```

Ingress 控制器配置参数

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

提示

您还可以应用以下 YAML 来配置路由准入策略：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

9.9.12. 使用通配符路由

HAProxy Ingress Controller **支持通配符路由。**Ingress Operator **使用 wildcardPolicy 来配置** Ingress Controller **的 ROUTER_ALLOW_WILDCARD_ROUTES 环境变量。**

Ingress Controller **的默认行为是接受采用 None 通配符策略的路由，该策略与现有 IngressController 资源向后兼容。**

流程

1. **配置通配符策略。**
   a. **使用以下命令来编辑 IngressController 资源：**

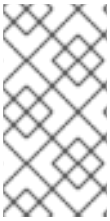
```
$ oc edit IngressController
```

   b. **在 spec 下，将 wildcardPolicy 字段设置为 WildcardsDisallowed 或 WildcardsAllowed：**

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
```

9.9.13. HTTP 标头配置

OpenShift Container Platform 提供了不同的使用 HTTP 标头的方法。在设置或删除标头时，您可以使用 Ingress Controller 中的特定字段或单独的路由来修改请求和响应标头。您还可以使用路由注解设置某些标头。配置标头的各种方法在协同工作时可能会带来挑战。



注意

您只能在 IngressController 或 Route CR 中设置或删除标头，您无法附加它们。如果使用值设置 HTTP 标头，则该值必须已完成，且在以后不需要附加。在附加标头（如 X-Forwarded-For 标头）时，请使用 spec.httpHeaders.forwardedHeaderPolicy 字段，而不是 spec.httpHeaders.actions。

9.9.13.1. 优先级顺序

当在 Ingress Controller 和路由中修改相同的 HTTP 标头时，HAProxy 会根据它是请求还是响应标头来优先选择操作。

- 对于 HTTP 响应标头，Ingress Controller 中指定的操作会在路由中指定的操作后执行。这意味着 Ingress Controller 中指定的操作具有优先权。
- 对于 HTTP 请求标头，路由中指定的操作会在 Ingress Controller 中指定的操作后执行。这意味着路由中指定的操作具有优先权。

例如，集群管理员使用以下配置设置 X-Frame-Options 响应标头，其值为 DENY ：

IngressController spec 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

路由所有者设置 Ingress Controller 中设置的相同响应标头，但使用以下配置值 SAMEORIGIN ：

Route 规格示例

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
```

```
      type: Set
      set:
        value: SAMEORIGIN
```

当 IngressController spec 和 Route spec 都配置 X-Frame-Options 标头时，Ingress Controller 中的全局级别为这个标头设置的值将具有优先权，即使一个特定的路由允许帧。对于请求标头，Route spec 值会覆盖 IngressController spec 值。

发生这个优先级，因为 haproxy.config 文件使用以下逻辑，其中 Ingress Controller 被视为前端，单独的路由被视为后端。应用到前端配置的标头值 DENY 使用后端中设置的值 SAMEORIGIN 覆盖相同的标头：

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

另外，Ingress Controller 或路由中定义的任何操作都覆盖使用路由注解设置的值。

9.9.13.2. 特殊情况标头

以下标头可能会阻止完全被设置或删除，或者在特定情况下允许：

表 9.2. 特殊情况标头配置选项

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
proxy	否	否	**proxy** HTTP 请求标头可以通过将标头值注入 **HTTP_PROXY** 环境变量来利用这个安全漏洞的 CGI 应用程序。**proxy** HTTP 请求标头也是非标准的，在配置期间容易出错。	否
主机	否	是	当使用 **IngressController** CR 设置 **host** HTTP 请求标头时，HAProxy 在查找正确的路由时可能会失败。	否

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
strict-transport-security	否	否	**strict-transport-security** HTTP 响应标头已使用路由注解处理，不需要单独的实现。	是：**haproxy.router.openshift.io/hsts_header** 路由注解
cookie 和 **set-cookie**	否	否	HAProxy 集的 Cookie 用于会话跟踪，用于将客户端连接映射到特定的后端服务器。允许设置这些标头可能会影响 HAProxy 的会话关联，并限制 HAProxy 的 Cookie 的所有权。	是： • **haproxy.router.openshift.io/disable_cookie** 路由注解 • **haproxy.router.openshift.io/cookie_name** 路由注解

9.9.14. 在 Ingress Controller 中设置或删除 HTTP 请求和响应标头

出于合规的原因，您可以设置或删除某些 HTTP 请求和响应标头。您可以为 Ingress Controller 提供的所有路由或特定路由设置或删除这些标头。

例如，您可能希望将集群中运行的应用程序迁移到使用 mutual TLS，这需要您的应用程序检查 X-Forwarded-Client-Cert 请求标头，但 OpenShift Container Platform 默认 Ingress Controller 提供了一个 X-SSL-Client-Der 请求标头。

以下流程修改 Ingress Controller 来设置 X-Forwarded-Client-Cert 请求标头，并删除 X-SSL-Client-Der 请求标头。

先决条件

- **已安装 OpenShift CLI（oc）。**
- **您可以使用具有 cluster-admin 角色的用户访问 OpenShift Container Platform 集群。**

流程

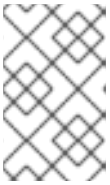
1. 编辑 Ingress Controller 资源：

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. **将 X-SSL-Client-Der HTTP 请求标头替换为 X-Forwarded-Client-Cert HTTP 请求标头：**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: 1
      request: 2
      - name: X-Forwarded-Client-Cert 3
        action:
          type: Set 4
          set:
            value: "%{+Q}[ssl_c_der,base64]" 5
      - name: X-SSL-Client-Der
        action:
          type: Delete
```

1. 要在 HTTP 标头上执行的操作列表。
2. 您要更改的标头类型。在本例中，请求标头。
3. 您要更改的标头的名称。有关您可以设置或删除的可用标头列表，请参阅 *HTTP 标头配置*。
4. 在标头中执行的操作类型。此字段可以具有 Set 或 Delete 的值。
5. 在设置 HTTP 标头时，您必须提供一个 value。该值可以是该标头的可用指令列表中的字符串，如 DENY，也可以是使用 HAProxy 的动态值语法来解释的动态值。在这种情况下，会添加一个动态值。



注意

对于 HTTP 响应设置动态标头值，允许示例 fetchers 是 res.hdr 和 ssl_c_der。对于 HTTP 请求设置动态标头值，允许示例获取器为 req.hdr 和 ssl_c_der。请求和响应动态值都可以使用 lower 和 base64 转换器。

3. 保存文件以使改变生效。

9.9.15. 使用 X-Forwarded 标头

您可以将 HAProxy Ingress Controller 配置为指定如何处理 HTTP 标头的策略，其中包括 Forwarded 和 X-Forwarded-For。Ingress Operator 使用 HTTPHeaders 字段配置 Ingress Controller 的 ROUTER_SET_FORWARDED_HEADERS 环境变量。

流程

1. 为 Ingress Controller 配置 HTTPHeaders 字段。
   a. 使用以下命令来编辑 IngressController 资源：

   ```
   $ oc edit IngressController
   ```

   b. 在 spec 下，将 HTTPHeaders 策略字段设置为 Append、Replace、IfNone 或 Never：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

使用案例示例
作为集群管理员，您可以：

- 配置将 X-Forwarded-For 标头注入每个请求的外部代理，然后将其转发到 Ingress Controller。
  要将 Ingress Controller 配置为通过未修改的标头传递，您需要指定 never 策略。然后，Ingress Controller 不会设置标头，应用程序只接收外部代理提供的标头。

- 将 Ingress Controller 配置为通过未修改的外部代理在外部集群请求上设置 X-Forwarded-For 标头。
  要将 Ingress Controller 配置为在不通过外部代理的内部集群请求上设置 X-Forwarded-For 标头，请指定 if-none 策略。如果 HTTP 请求已经通过外部代理设置了标头，则 Ingress Controller 会保留它。如果缺少标头，因为请求没有通过代理，Ingress Controller 会添加标头。

作为应用程序开发人员，您可以：

- 配置特定于应用程序的外部代理来注入 X-Forwarded-For 标头。
  要配置 Ingress Controller，以便在不影响其他路由策略的情况下将标头传递到应用程序的路由，请在应用程序的路由上添加注解 haproxy.router.openshift.io/set-forwarded-headers: if-none 或 haproxy.router.openshift.io/set-forwarded-headers: never。

注意

您可以根据每个路由设置 haproxy.router.openshift.io/set-forwarded-headers 注解，独立于 Ingress Controller 的全局设置值。

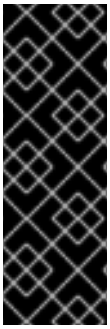
9.9.16. 启用 HTTP/2 入口连接

您可以在 HAProxy 中启用透明端到端的 HTTP/2 连接。此功能使应用程序所有者利用 HTTP/2 协议功能，包括单一连接、标头压缩、二 进制流等等。

您可以为单独的 Ingress Controller 或整个集群启用 HTTP/2 连接。

要在从客户端到 HAProxy 的连接中启用 HTTP/2，路由必须指定一个自定义证书。使用默认证书的路由无法使用 HTTP/2。这一限制是避免连接并发问题（如客户端为使用相同证书的不同路由重新使用连接）所必需的。

从 HAProxy 到应用程序 pod 的连接只能将 HTTP/2 用于 re-encrypt 路由，而不适用于 edge-terminated 或 insecure 路由。存在这个限制的原因是，在与后端协商使用 HTTP/2 时，HAProxy 要使用 ALPN（Application-Level Protocol Negotiation），它是一个 TLS 的扩展。这意味着，端到端的 HTTP/2 适用于 passthrough 和 re-encrypt 路由，而不适用于 nsecure 或 edge-terminated 路由。



重要

对于非 passthrough 路由，Ingress Controller 会独立于客户端的连接来协商它与应用程序的连接。这意味着，客户端可以连接到 Ingress Controller 并协商 HTTP/1.1，Ingress Controller 可连接到应用程序，协商 HTTP/2 并使用 HTTP/2 连接将客户端 HTTP/1.1 连接转发请求。如果客户端随后试图将其连接从 HTTP/1.1 升级到 WebSocket 协议，这会导致问题。因为 Ingress Controller 无法将 WebSocket 转发到 HTTP/2，也无法将其 HTTP/2 的连接升级到 WebSocket。因此，如果您有一个应用程序旨在接受 WebSocket 连接，则必须允许使用 HTTP/2 协议，或者其它客户端将无法升级到 WebSocket 协议。

流程

在单一 Ingress Controller 上启用 HTTP/2。

- 要在 Ingress Controller 上启用 HTTP/2，请输入 **oc annotate** 命令：

```
$ oc -n openshift-ingress-operator annotate
ingresscontrollers/<ingresscontroller_name> ingress.operator.openshift.io/default-
enable-http2=true
```

将 **<ingresscontroller_name>** 替换为要注解的 Ingress Controller 的名称。

在整个集群中启用 HTTP/2。

- 要为整个集群启用 HTTP/2，请输入 **oc annotate** 命令：

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-
http2=true
```

提示

您还可以应用以下 YAML 来添加注解：

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "true"
```

9.9.17. 为 Ingress Controller 配置 PROXY 协议

当 Ingress Controller 使用 **HostNetwork** 或 **NodePortService** 端点发布策略类型时，集群管理员可配置 PROXY 协议。PROXY 协议使负载均衡器能够为 Ingress Controller 接收的连接保留原始客户端地址。原始客户端地址可用于记录、过滤和注入 HTTP 标头。在默认配置中，Ingress Controller 接收的连接只包含与负载均衡器关联的源地址。

云部署不支持此功能。具有这个限制的原因是，当 OpenShift Container Platform 在云平台中运行时，IngressController 指定应使用服务负载均衡器，Ingress Operator 会配置负载均衡器服务，并根据保留源地址的平台要求启用 PROXY 协议。

重要

您必须将 OpenShift Container Platform 和外部负载均衡器配置为使用 PROXY 协议或使用 TCP。

警告

在使用 Keepalived Ingress VIP 的非云平台上带有安装程序置备的集群的默认 Ingress Controller 不支持 PROXY 协议。

先决条件

- 已创建一个 Ingress Controller。

流程

1. 编辑 Ingress Controller 资源：

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. 设置 PROXY 配置：

- 如果您的 Ingress Controller 使用 hostNetwork 端点发布策略类型，将 spec.endpointPublishingStrategy.hostNetwork.protocol 子字段设置为 PROXY：

  hostNetwork 配置为 PROXY 的示例

```
spec:
  endpointPublishingStrategy:
    hostNetwork:
      protocol: PROXY
    type: HostNetwork
```

- 如果您的 Ingress Controller 使用 NodePortService 端点发布策略类型，将 spec.endpointPublishingStrategy.nodePort.protocol 子字段设置为 PROXY：

  nodePort 配置为 PROXY 示例

```
spec:
  endpointPublishingStrategy:
    nodePort:
      protocol: PROXY
    type: NodePortService
```

9.9.18. 使用 appsDomain 选项指定备选集群域

作为集群管理员，您可以通过配置 appsDomain 字段来为用户创建的路由指定默认集群域替代内容。appsDomain 字段是 OpenShift Container Platform 使用的可选域，而不是默认值，它在domain 字段中指定。如果您指定了其它域，它会覆盖为新路由确定默认主机的目的。

例如，您可以将您公司的 DNS 域用作集群中运行的应用程序的路由和入口的默认域。

先决条件

- 已部署 OpenShift Container Platform 集群。
- 已安装 oc 命令行界面。

流程

1. 通过为用户创建的路由指定备选默认域来配置 appsDomain 字段。
   a. 编辑 ingress 集群资源 ：

```
$ oc edit ingresses.config/cluster -o yaml
```

   b. 编辑 YAML 文件：

   示例 appsDomain 配置为 test.example.com

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.example.com                    (1)
  appsDomain: <test.example.com>              (2)
```

   (1) 指定默认域。您不能在安装后修改默认域。

   (2) 可选：用于应用程序路由的 OpenShift Container Platform 基础架构域。您可以使用测试 等替代前缀 apps，而不是默认前缀。

2. 通过公开路由并验证路由域更改，验证现有路由是否包含 appsDomain 字段中指定的域名：

   注意

   在公开路由前，等待 openshift-apiserver 完成滚动更新。

   a. 公开路由：

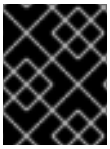
```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

   输出示例：

```
$ oc get routes
NAME              HOST/PORT                                   PATH   SERVICES         PORT
TERMINATION   WILDCARD
hello-openshift   hello_openshift-<my_project>.test.example.com
hello-openshift   8080-tcp                None
```

9.9.19. 转换 HTTP 标头的大小写

默认情况下，HAProxy HTTP 的标头名称是小写的，例如，会将 **Host: xyz.com** 更改为 **host: xyz.com**。如果旧应用程序对 HTTP 标头名称中使用大小写敏感，请使用 Ingress Controller **spec.httpHeaders.headerNameCaseAdjustments** API 字段进行调整来适应旧的应用程序，直到它们被改变。



重要

由于 OpenShift Container Platform 包含 HAProxy 2.8，因此请确保在升级前使用 **spec.httpHeaders.headerNameCaseAdjustments** 添加必要的配置。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

作为集群管理员，您可以使用 **oc patch** 命令，或设置 Ingress Controller YAML 文件中的 **HeaderNameCaseAdjustments** 字段来转换 HTTP 标头的大小写。

- 使用 **oc patch** 命令设置一个 HTTP 标头的大小写情况。
  1. 输入 **oc patch** 命令将 HTTP **host** 标头改为 **Host**：

     ```
     $ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}}'
     ```

  2. 注解应用程序的路由：

     ```
     $ oc annotate routes/my-application haproxy.router.openshift.io/h1-adjust-case=true
     ```

     然后，Ingress Controller 会根据指定调整 **host** 请求标头。

- 通过配置 Ingress Controller YAML 文件，使用 **HeaderNameCaseAdjustments** 字段指定调整。
  1. 以下 Ingress Controller YAML 示例将 HTTP/1 请求的 **host** 标头调整为 **Host**，以便可以适当地注解路由：

     Ingress Controller YAML 示例

     ```
     apiVersion: operator.openshift.io/v1
     kind: IngressController
     metadata:
       name: default
       namespace: openshift-ingress-operator
     spec:
       httpHeaders:
         headerNameCaseAdjustments:
         - Host
     ```

2. 以下示例路由中，使用 haproxy.router.openshift.io/h1-adjust-case 注解启用对 HTTP 响应标头名称的大小写调整：

   路由 YAML 示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

(1) 将 haproxy.router.openshift.io/h1-adjust-case 设置为 true。

9.9.20. 使用路由器压缩

您可以将 HAProxy Ingress Controller 配置为为特定 MIME 类型全局指定路由器压缩。您可以使用 mimeTypes 变量定义压缩应用到的 MIME 类型的格式。类型包括：application, image, message, multipart, text, video, 或带有一个 "X-" 前缀的自定义类型。要查看 MIME 类型和子类型的完整表示法，请参阅 RFC1341。

注意

为压缩分配的内存可能会影响最大连接。此外，对大型缓冲区的压缩可能导致延迟，如非常复杂的正则表达式或较长的正则表达式列表。

并非所有 MIME 类型从压缩中受益，但 HAProxy 仍然使用资源在指示时尝试压缩。通常而言，文本格式（如 html、css 和 js）与压缩格式获益，但已经压缩的格式（如图像、音频和视频）可能会因为需要压缩操作而无法获得太多的好处。

流程

1. 为 Ingress Controller 配置 httpCompression 字段。

   a. 使用以下命令来编辑 IngressController 资源：

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```

   b. 在 spec 下，将 httpCompression 策略字段设置为 mimeTypes，并指定应该应用压缩的 MIME 类型列表：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
```

```
  mimeTypes:
  - "text/html"
  - "text/css; charset=utf-8"
  - "application/json"
  ...
```

9.9.21. 公开路由器指标

您可以在默认统计端口 1936 上以 Prometheus 格式公开 HAProxy 路由器指标。外部指标收集和聚合系统（如 Prometheus）可以访问 HAProxy 路由器指标。您可以在浏览器中以 HTML 的形式和以逗号分隔的值 (CSV) 格式查看 HAProxy 路由器指标。

先决条件

- 您已将防火墙配置为访问默认统计数据端口 1936。

流程

1. 运行以下命令来获取路由器 pod 名称：

```
$ oc get pods -n openshift-ingress
```

   输出示例

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-76bfffb66c-46qwp   1/1     Running   0          11h
```

2. 获取路由器的用户名和密码，路由器 Pod 存储在 /var/lib/haproxy/conf/metrics-auth/statsUsername 和 /var/lib/haproxy/conf/metrics-auth/statsPassword 文件中：

   a. 运行以下命令来获取用户名：

```
$ oc rsh <router_pod_name> cat metrics-auth/statsUsername
```

   b. 运行以下命令来获取密码：

```
$ oc rsh <router_pod_name> cat metrics-auth/statsPassword
```

3. 运行以下命令，获取路由器 IP 和指标证书：

```
$ oc describe pod <router_pod>
```

4. 运行以下命令，以 Prometheus 格式获取原始统计信息：

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

5. 运行以下命令来安全地访问指标：

```
$ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
```

6. 运行以下命令，访问默认的 stats 端口 1936：

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

例 9.1. 输出示例

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route=
"hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route=
"hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route=
"hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the
current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",
service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",serv
ice=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-
registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-
vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. 通过在浏览器中输入以下 URL 来启动 stats 窗口：

```
http://<user>:<password>@<router_IP>:<stats_port>
```

8. 可选：通过在浏览器中输入以下 URL 来获取 CSV 格式的统计信息：

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

9.9.22. 自定义 HAProxy 错误代码响应页面

作为集群管理员，您可以为 503、404 或两个错误页面指定自定义错误代码响应页面。当应用 Pod 没有运行时，HAProxy 路由器会提供一个 503 错误页面，如果请求的 URL 不存在，则 HAProxy 路由器会提供 404 错误页面。例如，如果您自定义 503 错误代码响应页面，则应用 Pod 未运行时会提供页面，并且

HAProxy 路由器为不正确的路由或不存在的路由提供默认的 404 错误代码 HTTP 响应页面。

自定义错误代码响应页面在配置映射中指定，然后修补至 Ingress Controller。配置映射键有两个可用的文件名，如下所示：error-page-503.http 和 error-page-404.http。

自定义 HTTP 错误代码响应页面必须遵循 HAProxy HTTP 错误页面配置指南。以下是默认 OpenShift Container Platform HAProxy 路由器 http 503 错误代码响应页面的示例。您可以使用默认内容作为模板来创建自己的自定义页面。

默认情况下，当应用没有运行或者路由不正确或不存在时，HAProxy 路由器仅提供一个 503 错误页面。此默认行为与 OpenShift Container Platform 4.8 及更早版本中的行为相同。如果没有提供用于自定义 HTTP 错误代码响应的配置映射，且您使用的是自定义 HTTP 错误代码响应页面，路由器会提供默认的 404 或 503 错误代码响应页面。

注意

如果您使用 OpenShift Container Platform 默认 503 错误代码页面作为自定义的模板，文件中的标头需要编辑器而不是使用 CRLF 行结尾。

流程

1. 在 openshift-config 命名空间中创建一个名为 my-custom-error-code-pages 的配置映射：

```
$ oc -n openshift-config create configmap my-custom-error-code-pages \
--from-file=error-page-503.http \
--from-file=error-page-404.http
```

重要

如果没有为自定义错误代码响应页面指定正确的格式，则会出现路由器 pod 中断。要解决此中断，您必须删除或更正配置映射并删除受影响的路由器 pod，以便使用正确的信息重新创建它们。

2. 对 Ingress Controller 进行补丁以根据名称引用 my-custom-error-code-pages 配置映射：

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":
{"httpErrorCodePages":{"name":"my-custom-error-code-pages"}}}' --type=merge
```

Ingress Operator 将 my-custom-error-code-pages 配置映射从 openshift-config 命名空间复制到 openshift-ingress 命名空间。Operator 根据 openshift-ingress 命名空间中的模式 <your_ingresscontroller_name>-errorpages 命名配置映射。

3. 显示副本：

```
$ oc get cm default-errorpages -n openshift-ingress
```

输出示例

```
NAME                 DATA   AGE
default-errorpages     2    25s  1
```

1. 配置映射名称示例为 default-errorpages，因为 default Ingress Controller 自定义资源 (CR) 已被修补。

4. 确认包含自定义错误响应页面的配置映射挂载到路由器卷中，其中配置映射键是具有自定义 HTTP 错误代码响应的文件名：

   - 对于 503 自定义 HTTP 自定义错误代码响应：

     ```
     $ oc -n openshift-ingress rsh <router_pod> cat
     /var/lib/haproxy/conf/error_code_pages/error-page-503.http
     ```

   - 对于 404 自定义 HTTP 自定义错误代码响应：

     ```
     $ oc -n openshift-ingress rsh <router_pod> cat
     /var/lib/haproxy/conf/error_code_pages/error-page-404.http
     ```

验证

验证自定义错误代码 HTTP 响应：

1. 创建测试项目和应用程序：

   ```
   $ oc new-project test-ingress
   ```

   ```
   $ oc new-app django-psql-example
   ```

2. 对于 503 自定义 http 错误代码响应：

   a. 停止应用的所有容器集。

   b. 运行以下 curl 命令或在浏览器中访问路由主机名：

      ```
      $ curl -vk <route_hostname>
      ```

3. 对于 404 自定义 http 错误代码响应：

   a. 访问不存在的路由或路由不正确。

   b. 运行以下 curl 命令或在浏览器中访问路由主机名：

      ```
      $ curl -vk <route_hostname>
      ```

4. 检查 haproxy.config 文件中的 errorfile 属性是否正确：

   ```
   $ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep
   errorfile
   ```

9.9.23. 设置 Ingress Controller 最大连接数

集群管理员可以设置 OpenShift 路由器部署的最大同时连接数。您可以修补现有的 Ingress Controller 来提高最大连接数。

先决条件

- 以下假设您已创建了 Ingress Controller

流程

- 更新 Ingress Controller，以更改 HAProxy 的最大连接数：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p
'{"spec":{"tuningOptions": {"maxConnections": 7500}}}'
```

警告

如果您设置了大于当前操作系统的 spec.tuningOptions.maxConnections 值，则 HAProxy 进程不会启动。有关这个参数的更多信息，请参阅"Ingress Controller 配置参数"部分中的表。

9.10. 其他资源

- 配置自定义 PKI

第 10 章 OPENSHIFT CONTAINER PLATFORM 中的 INGRESS 分片

在 OpenShift Container Platform 中，Ingress Controller 可以服务所有路由，也可以提供路由的子集。默认情况下，Ingress Controller 提供集群中任何命名空间中创建的任何路由。您可以在集群中添加额外的 Ingress Controller，以通过创建 *分片*来优化路由，这些分片是基于所选特征的路由子集。要将路由标记为分片的成员，请使用 route 或 namespace **metadata** 字段中的标签。Ingress Controller 使用*选择器*（也称为 *选择表达式*）从要提供服务的整个路由池中选择路由子集。

当您希望在多个 Ingress Controller 之间负载平衡传入的流量时，当您要隔离到特定 Ingress Controller 的流量或下一部分中描述的各种其他原因时，Ingress 分片很有用。

默认情况下，每个路由都使用集群的默认域。但是，可以将路由配置为使用路由器的域。如需更多信息，请参阅为 Ingress Controller 创建路由。

10.1. INGRESS CONTROLLER 分片

您可以通过向路由、命名空间或两者添加标签，使用 Ingress 分片（也称为路由器分片）在多个路由器之间分发一组路由。Ingress Controller 使用一组对应的选择器来只接受具有指定标签的路由。每个 Ingress 分片都由使用给定选择表达式过滤的路由组成。

Ingress Controller 是网络流量进入集群的主要机制，因此对它们的需求可能非常大。作为集群管理员，您可以对路由进行分片，以达到以下目的：

- 在 Ingress Controller 或路由器与一些路由之间实现平衡，由此加快对变更的响应。
- 分配特定的路由，使其具有不同于其它路由的可靠性保证。
- 允许特定的 Ingress Controller 定义不同的策略。
- 只允许特定的路由使用其他功能。
- 在不同的地址上公开不同的路由，例如使内部和外部用户能够看到不同的路由。
- 在蓝绿部署期间，将流量从应用的一个版本转移到另一个版本。

当 Ingress Controller 被分片时，一个给定路由被接受到组中的零个或多个 Ingress Controller。路由的状态描述了 Ingress Controller 是否已接受它。只有 Ingress Controller 对其分片是唯一的时，才会接受路由。

Ingress Controller 可以使用三个分片方法：

- 仅将命名空间选择器添加到 Ingress Controller，以便命名空间中带有与命名空间选择器匹配的标签的所有路由都位于 Ingress shard 中。
- 只向 Ingress Controller 添加路由选择器，因此所有与路由选择器匹配的标签的路由都位于 Ingress 分片中。
- 将命名空间选择器和路由选择器添加到 Ingress Controller 中，以便使用与命名空间选择器匹配的路由选择器匹配的标签的路由位于 Ingress shard 中。

使用分片，您可以在多个 Ingress Controller 上分发路由子集。这些子集可以是非重叠的，也称为 *传统* 分片，或是重叠的，也称为 *overlapped* 分片。

10.1.1. 传统分片示例

Ingress Controller **finops-router** 使用标签选择器 **spec.namespaceSelector.matchLabels.name** 设置为 **finance** 和 **ops** ：

finops-router 的 YAML 定义示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: finops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name:
        - finance
        - ops
```

第二个 Ingress Controller **dev-router** 配置有标签选择器 **spec.namespaceSelector.matchLabels.name** 设置为 **dev** ：

dev-router 的 YAML 定义示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: dev-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name: dev
```

如果所有应用程序路由都位于单独的命名空间中，每个命名空间都分别使用 **name:finance**、**name:ops** 和 **name:dev** 标记，此配置会在两个 Ingress Controller 之间有效分发您的路由。不应处理用于控制台、身份验证和其他目的的 OpenShift Container Platform 路由。

在上面的场景中，分片成为分区的一种特殊情况，没有重叠的子集。路由在路由器分片之间划分。

警告

默认 Ingress Controller 继续提供所有路由，除非**namespaceSelector** 或 **routeSelector** 字段包含用于排除的路由。有关如何从默认 Ingress Controller 中排除路由的更多信息，请参阅这个 红帽知识库解决方案 和"分片默认 Ingress Controller"。

10.1.2. 重叠的分片示例

除了上例中的 **finops-router** 和 **dev-router** 外，您也具有 **devops-router**，它被配置为标签选择器 **spec.namespaceSelector.matchLabels.name**，设置为 **dev** 和 **ops** ：

devops-router 的 YAML 定义示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: devops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name:
        - dev
        - ops
```

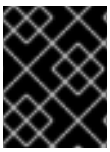
标签为 **name:dev 和 name:ops 的命名空间中的路由现在由两个不同的** Ingress Controller **服务。使用**这个**配置，您有重叠的路由子集。**

通过重叠的路由子集，您可以创建更复杂**的路由规则。例如，您可以在向 devops-router** 发送较低优先级**的流量时，将**优先级更高的流量放入专用的 finops-router。**

10.1.3. 分片默认 Ingress Controller

创建新的 Ingress shard **后，可能会接受到默认** Ingress Controller **接受的新** Ingress **分片的路由。**这是因为**默认** Ingress Controller **没有**选择器，**并默认**接受所有路由。

您可以使用命名空间选择器或路由选择器来限制 Ingress Controller **使用特定**标签**提供路由。以下流程限制默认** Ingress Controller，**使用命名空间选择器为新分片的 finance、ops 和 dev 提供。**这为 Ingress **分片增加了额外的隔离。**



重要

您必须在同一 Ingress Controller **上保留所有** OpenShift Container Platform **管理路由。因此，避免在排除**这些**基本路由的默认** Ingress Controller **中添加额外的**选择器。

先决条件

- **已安装** OpenShift CLI（**oc**）。
- **您以**项目管理员**身份登录。**

流程

1. **运行以下命令来修改默认** Ingress Controller：

```
$ oc edit ingresscontroller -n openshift-ingress-operator default
```

2. **编辑** Ingress Controller **以包含一个 namespaceSelector，它排除了任何 finance、ops 和 dev** 标签**的路由：**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
```

```
spec:
  namespaceSelector:
    matchExpressions:
      - key: type
        operator: NotIn
        values:
          - finance
          - ops
          - dev
```

默认 Ingress Controller 不再提供标记为 **name:finance**、**name:ops** 和 **name:dev** 的命名空间。

10.1.4. Ingress 分片和 DNS

集群管理员负责为项目中的每个路由器生成单独的 DNS 条目。路由器不会将未知路由转发到另一个路由器。

考虑以下示例：

- 路由器 A 驻留在主机 192.168.0.5 上，并且具有 ***.foo.com** 的路由。
- 路由器 B 驻留在主机 192.168.1.9 上，并且具有 ***.example.com** 的路由。

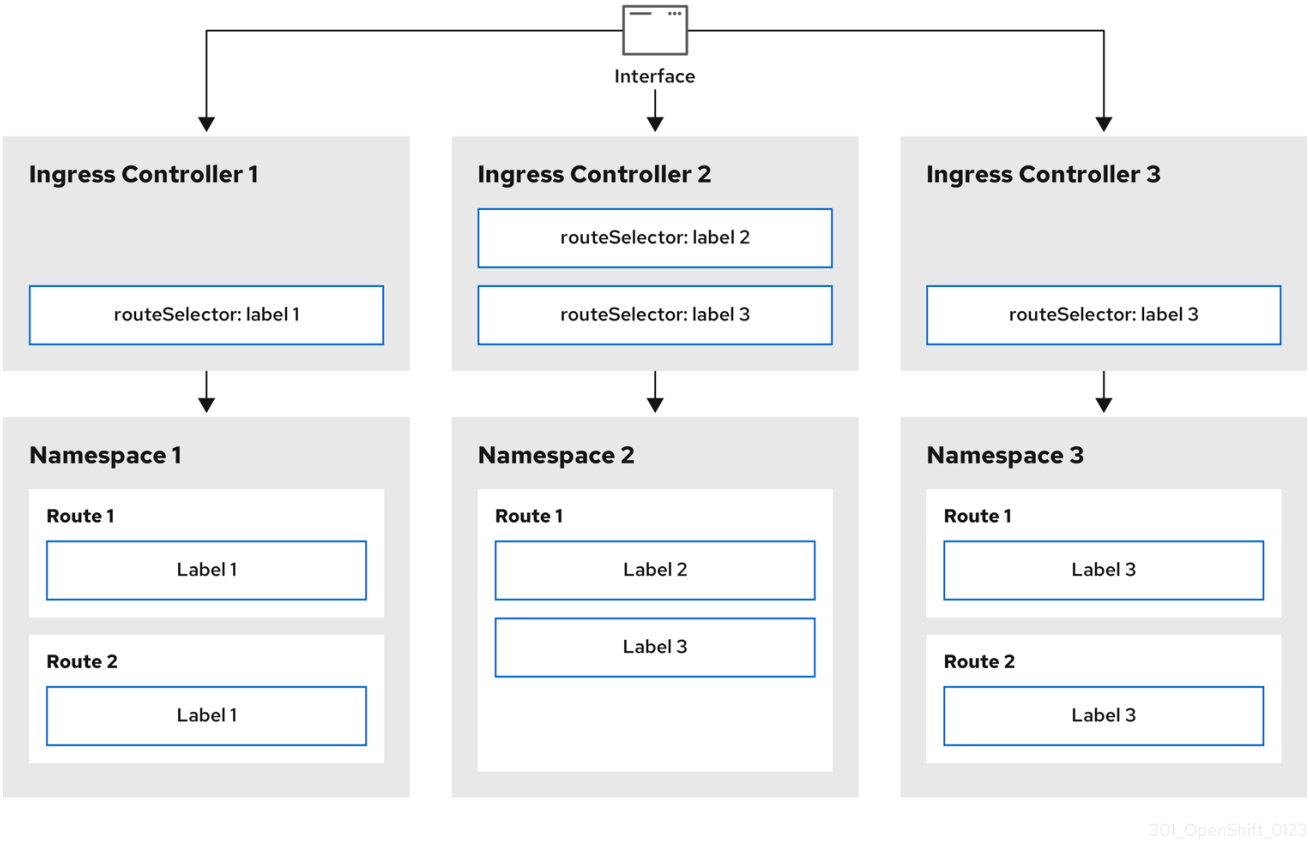
单独的 DNS 条目必须将 ***.foo.com** 解析为托管 Router A 和 ***.example.com** 的节点到托管路由器 B 的节点：

- ***.foo.com A IN 192.168.0.5**
- ***.example.com A IN 192.168.1.9**

10.1.5. 通过路由标签（label）配置 Ingress Controller 分片

使用路由标签进行 Ingress Controller 分片，意味着 Ingress Controller 提供由路由选择器选择的任意命名空间中的所有路由。

图 10.1. 使用路由标签进行 Ingress 分片



在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

流程

1. 编辑 **router-internal.yaml** 文件：

```
# cat router-internal.yaml
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  routeSelector:
    matchLabels:
      type: sharded
```

1 指定 Ingress Controller 使用的域。此域必须与默认 Ingress Controller 域不同。

2. 应用 Ingress Controller **router-internal.yaml** 文件：

```
# oc apply -f router-internal.yaml
```

Ingress Controller 选择具有 type: sharded 标签的任意命名空间中的路由。

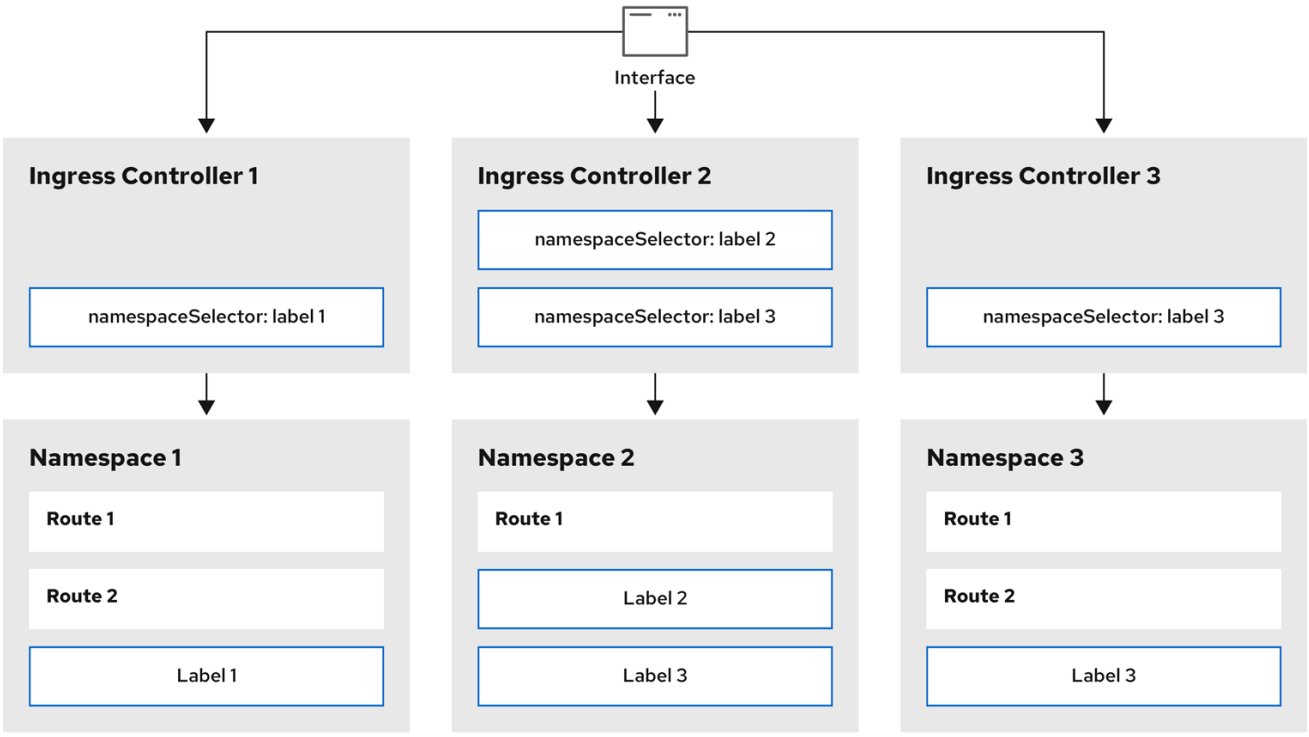
3. 使用 router-internal.yaml 中配置的域创建新路由：

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

10.1.6. 使用命名空间标签配置 Ingress Controller 分片

使用命名空间标签进行 Ingress Controller 分片，意味着 Ingress Controller 提供由命名空间选择器选择的任意命名空间中的所有路由。

图 10.2. 使用命名空间标签进行 Ingress 分片



在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

流程

1. 编辑 router-internal.yaml 文件：

```
# cat router-internal.yaml
```

输出示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
```

```
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> (1)
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

(1) 指定 Ingress Controller 使用的域。此域必须与默认 Ingress Controller 域不同。

2. 应用 Ingress Controller **router-internal.yaml** 文件：

```
# oc apply -f router-internal.yaml
```

Ingress Controller 选择由命名空间选择器选择的具有 **type: sharded** 标签的任意命名空间中的路由。

3. 使用 **router-internal.yaml** 中配置的域创建新路由：

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

10.2. 为 INGRESS CONTROLLER 分片创建路由

通过使用路由，您可以通过 URL 托管应用程序。在这种情况下，主机名没有被设置，路由会使用子域。当您指定子域时，会自动使用公开路由的 Ingress Controller 域。对于由多个 Ingress Controller 公开路由的情况，路由由多个 URL 托管。

以下流程描述了如何为 Ingress Controller 分片创建路由，使用 **hello-openshift** 应用程序作为示例。

在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您以项目管理员身份登录。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。
- 您已为分片配置了 Ingress Controller。

流程

1. 运行以下命令，创建一个名为 **hello-openshift** 的项目：

```
$ oc new-project hello-openshift
```

2. 运行以下命令，在项目中创建 pod：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-
openshift/hello-pod.json
```

3. 运行以下命令，创建名为 hello-openshift 的服务：

```
$ oc expose pod/hello-openshift
```

4. 创建名为 hello-openshift-route.yaml 的路由定义：

为分片创建的路由的 YAML 定义：

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

(1) 标签键及其对应标签值必须与 Ingress Controller 中指定的标签值匹配。在本例中，Ingress Controller 具有标签键和值 type: sharded。

(2) 路由将使用 subdomain 字段的值公开。指定 subdomain 字段时，您必须保留主机名未设置。如果您同时指定了 host 和 subdomain 字段，则路由将使用 host 字段的值，并忽略 subdomain 字段。

5. 通过运行以下命令，使用 hello-openshift-route.yaml 创建到 hello-openshift 应用程序的路由：

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

验证

- 使用以下命令获取路由的状态：

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

生成的 Route 资源应类似以下示例：

输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-
sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1 Ingress Controller 或路由器的主机名用于公开路由。host 字段的值由 Ingress Controller 自动决定，并使用它的域。在本例中，Ingress Controller 的域为 <apps-sharded.basedomain.example.net>。

2 Ingress Controller 的主机名。

3 Ingress Controller 的名称。在本例中，Ingress Controller 的名称为 sharded。

其它资源

- Ingress Controller（router）性能的基线

第 11 章 为手动 DNS MANAGEMENT 配置 INGRESS CONTROLLER

作为集群管理员，在创建 Ingress Controller 时，Operator 会自动管理 DNS 记录。当所需的 DNS 区域与集群 DNS 区域不同或 DNS 区域被托管在云供应商时，这有一些限制。

作为集群管理员，您可以将 Ingress Controller 配置为停止自动 DNS 管理并启动手动 DNS 管理。将 **dnsManagementPolicy** 设置为指定应自动或手动管理的时间。

当您将 Ingress Controller 从 **Managed** 改为 **Unmanaged** DNS 管理策略时，Operator 不会清理在云中置备的以前的通配符 DNS 记录。当您将 Ingress Controller 从 **Unmanaged** 改为 **Managed** DNS 管理策略时，Operator 会尝试在云供应商上创建 DNS 记录（如果不存在），或更新 DNS 记录（如果已存在）。

重要

当您将 **dnsManagementPolicy** 设置为 **unmanaged** 时，您必须手动管理云供应商上的通配符 DNS 记录的生命周期。

11.1. MANAGED DNS 管理策略

Ingress Controller 的 **Managed** DNS 管理策略可确保云供应商上通配符 DNS 记录的生命周期由 Operator 自动管理。

11.2. UNMANAGED DNS 管理策略

Ingress Controller 的 **Unmanaged** DNS 管理策略可确保云供应商上通配符 DNS 记录的生命周期不会自动管理，而是由集群管理员负责。

注意

在 AWS 云平台中，如果 Ingress Controller 上的域与 **dnsConfig.Spec.BaseDomain** 不匹配，则 DNS 管理策略会自动设置为 **Unmanaged**。

11.3. 使用 UNMANAGED DNS 管理策略创建自定义 INGRESS CONTROLLER

作为集群管理员，您可以使用 **Unmanaged** DNS 管理策略创建新的自定义 Ingress Controller。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建名为 **sample-ingress.yaml** 的自定义资源 (CR) 文件，包含以下内容：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
```

```
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External 3
      dnsManagementPolicy: Unmanaged 4
```

1. 使用 IngressController 对象的名称指定 <name>。
2. 根据作为前提条件创建的 DNS 记录指定 domain。
3. 将 scope 指定为 External，以在外部公开负载均衡器。
4. dnsManagementPolicy 表示 Ingress Controller 是否管理与负载均衡器关联的通配符 DNS 记录的生命周期。有效值为 Managed 和 Unmanaged。默认值为 Managed。

2. 保存文件以使改变生效。

```
oc apply -f <name>.yaml 1
```

11.4. 修改现有 INGRESS CONTROLLER

作为集群管理员，您可以修改现有 Ingress Controller 以手动管理 DNS 记录生命周期。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 修改所选 IngressController 来设置 dnsManagementPolicy ：

```
SCOPE=$(oc -n openshift-ingress-operator get ingresscontroller <name> -
o=jsonpath="{.status.endpointPublishingStrategy.loadBalancer.scope}")

oc -n openshift-ingress-operator patch ingresscontrollers/<name> --type=merge --
patch='{"spec":{"endpointPublishingStrategy":
{"type":"LoadBalancerService","loadBalancer":
{"dnsManagementPolicy":"Unmanaged", "scope":"${SCOPE}"}}}}'
```

2. 可选：您可以删除云供应商中的关联的 DNS 记录。

11.5. 其他资源

- Ingress Controller 配置参数

第 12 章 配置 INGRESS CONTROLLER 端点发布策略

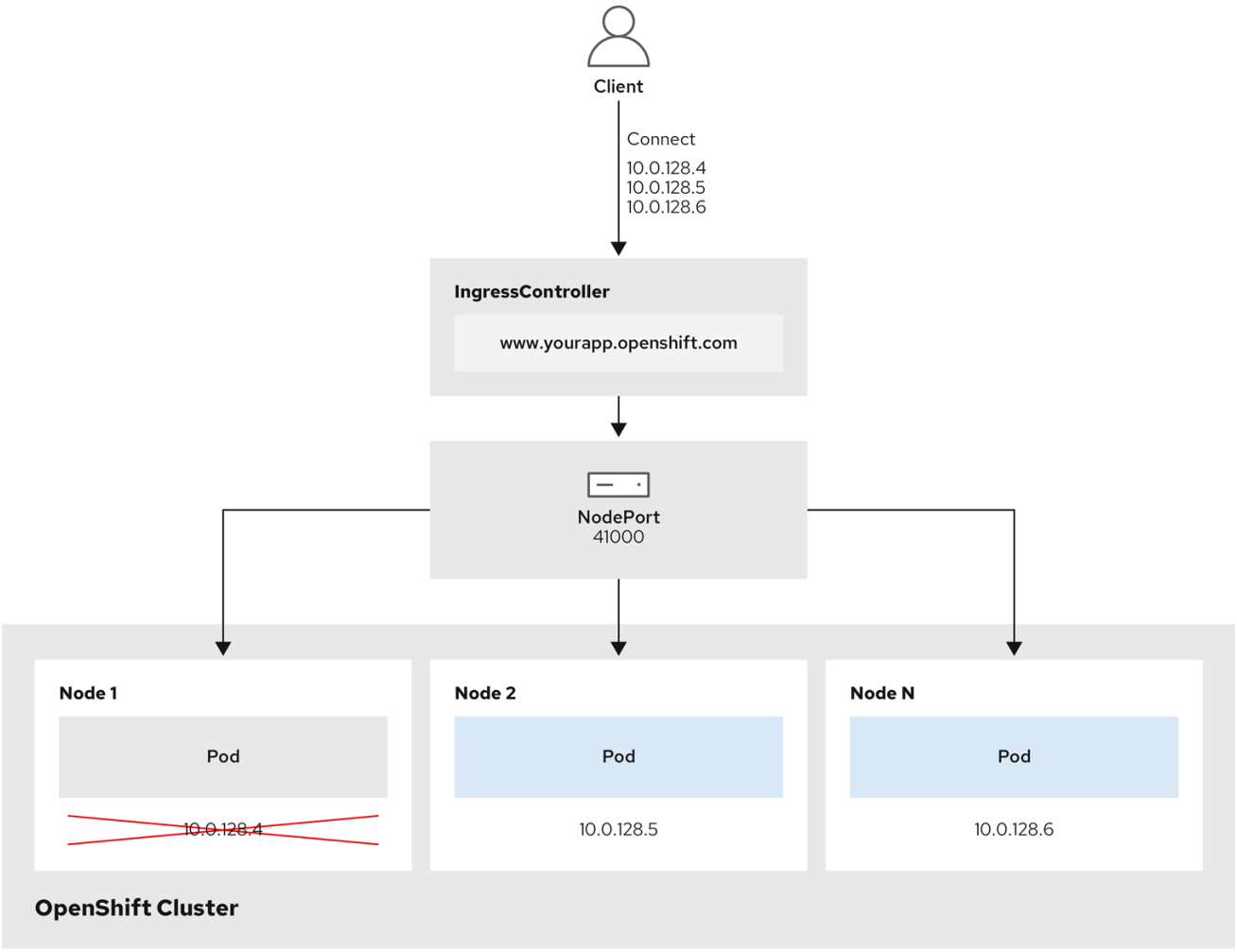
12.1. INGRESS CONTROLLER 端点发布策略

NodePortService 端点发布策略

NodePortService 端点发布策略使用 Kubernetes NodePort 服务发布 Ingress Controller。

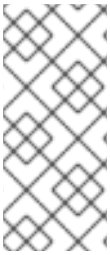
在这个配置中，Ingress Controller 部署使用容器网络。创建了一个 NodePortService 来发布部署。特定的节点端口由 OpenShift Container Platform 动态分配; 但是，为了支持静态端口分配，您会保留对受管 NodePortService 的节点端口字段的更改 。

图 12.1. NodePortService 图表



上图显示了与 OpenShift Container Platform Ingress NodePort 端点发布策略相关的以下概念：

- **集群中的所有可用节点均有自己的外部可访问 IP 地址。集群中运行的服务绑定到所有节点的唯一 NodePort。**
- **当客户端连接到停机的节点时，例如，通过连接图形中的 10.0.128.4 IP 地址，节点端口将客户端直接连接到运行该服务的可用节点。在这种情况下，不需要负载平衡。如图形中所显，10.0.128.4 地址已不可用，必须使用另一个 IP 地址。**



注意

Ingress Operator 忽略对服务的 **.spec.ports[].nodePort** 字段的任何更新。

默认情况下，端口会自动分配，您可以访问集成的端口分配。但是，有时需要静态分配端口来与现有基础架构集成，这些基础架构可能无法根据动态端口进行重新配置。要实现与静态节点端口的集成，您可以直接更新受管服务资源。

如需有关 daemonset 的更多信息，请参阅关于 NodePort 的 Kubernetes 服务文档。

HostNetwork 端点发布策略

HostNetwork 端点发布策略会在部署 Ingress Controller 的节点端口上发布 Ingress Controller。

带有 **HostNetwork** 端点发布策略的 Ingress Controller 每个节点只能有一个 pod 副本。如果您想要*n* 个副本，则必须至少使用可调度这些副本的 *n* 个节点。因为每个 Pod 副本都会通过调度的节点主机上的端口 **80** 和 **443** 进行请求，所以如果同一节点上的其他 pod 使用这些端口，则无法将副本调度到该节点。

12.1.1. 将 Ingress Controller 端点发布范围配置为 Internal

当集群管理员在没有指定集群为私有的情况下安装新集群时，将默认 Ingress Controller 创建，并将 **scope** 设置为 **External**。集群管理员可以将 **External** 范围的 Ingress Controller 更改为**Internal**。

先决条件

- 已安装 **oc** CLI。

流程

- 要将 **External** 范围的 Ingress Controller 更改为**Internal**，请输入以下命令：

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":
{"type":"LoadBalancerService","loadBalancer":{"scope":"Internal"}}}}'
```

- 要检查 Ingress Controller 的状态，请输入以下命令：

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

  - **Progressing** 状态条件指示您必须执行进一步的操作。例如，状态条件可以通过输入以下命令来指示需要删除该服务：

```
$ oc -n openshift-ingress delete services/router-default
```

    如果删除了该服务，Ingress Operator 会重新创建为 **Internal**。

12.1.2. 配置 Ingress Controller 端点发布范围到外部

当集群管理员在没有指定集群为私有的情况下安装新集群时，将默认 Ingress Controller 创建，并将 **scope** 设置为 **External**。

Ingress Controller 的范围可以在安装过程中或之后配置为 **Internal**，集群管理员可以将 内部 Ingress Controller 更改为 **External**。

重要

在某些平台上，需要删除并重新创建服务。

更改范围可能会导致 Ingress 流量中断，这可能会持续几分钟。这适用于需要删除和重新创建服务的平台，因为流程可能会导致 OpenShift Container Platform 取消置备现有服务负载均衡器、置备一个新服务负载均衡器并更新 DNS。

先决条件

- 已安装 oc CLI。

流程

- 要将 内部范围的 Ingress Controller 更改为外部，请输入以下命令：

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/private --type=merge --
patch='{"spec":{"endpointPublishingStrategy":
{"type":"LoadBalancerService","loadBalancer":{"scope":"External"}}}}'
```

- 要检查 Ingress Controller 的状态，请输入以下命令：

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

  - Progressing 状态条件指示您必须执行进一步的操作。例如，状态条件可以通过输入以下命令来指示需要删除该服务：

```
$ oc -n openshift-ingress delete services/router-default
```

    如果删除了该服务，Ingress Operator 会重新创建为 External。

12.2. 其他资源

- 如需更多信息，请参阅 Ingress Controller 配置参数。

第 13 章 验证到端点的连接

Cluster Network Operator（CNO）运行一个控制器（连接检查控制器），用于在集群的资源间执行连接健康检查。通过查看健康检查的结果，您可以诊断连接问题或解决网络连接问题，将其作为您要调查的问题的原因。

13.1. 执行连接健康检查

要验证集群资源是否可以访问，请向以下集群 API 服务的每个服务都有一个 TCP 连接：

- Kubernetes API 服务器服务
- Kubernetes API 服务器端点
- OpenShift API 服务器服务
- OpenShift API 服务器端点
- 负载均衡器

要验证服务和服务端点是否可在集群中的每个节点上访问，请对以下每个目标都进行 TCP 连接：

- 健康检查目标服务
- 健康检查目标端点

13.2. 连接健康检查实现

在集群中，连接检查控制器或编配连接验证检查。连接测试的结果存储在 **openshift-network-diagnostics** 命名空间中的 **PodNetworkConnectivity** 对象中。连接测试会每分钟以并行方式执行。

Cluster Network Operator（CNO）将几个资源部署到集群，以发送和接收连接性健康检查：

健康检查源

此程序部署在一个由 **Deployment** 对象管理的单个 pod 副本集中。程序会消耗 **PodNetworkConnectivity** 对象，并连接到每个对象中指定的 **spec.targetEndpoint**。

健康检查目标

pod 作为集群中每个节点上的守护进程集的一部分部署。pod 侦听入站健康检查。在每个节点上存在这个 pod 可以测试到每个节点的连接。

您可以使用节点选择器配置在其上运行网络连接源和目标的节点。另外，您可以为源和目标 pod 指定允许的*容限*。配置在 **config.openshift.io/v1** API 组中的 **Network** API 的单例 **cluster** 自定义资源中定义。

Pod 调度在更新了配置后发生。因此，您必须在更新配置前应用要在选择器中使用的节点标签。更新网络连接后应用的标签将忽略 pod 放置。

请参考以下 YAML 中的默认配置：

连接源和目标 pod 的默认配置

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
```

```
spec:
  # ...
    networkDiagnostics: 1
      mode: "" 2
      sourcePlacement: {} 3
      targetPlacement: {} 4
```

1 1 指定网络诊断配置。如果没有指定值，或者指定了空对象，并在名为 cluster 的 network.operator.openshift.io 自定义资源中设置 spec.disableNetworkDiagnostics=true，则会禁用网络诊断。如果设置，这个值会覆盖 spec.disableNetworkDiagnostics=true。

2 指定诊断模式。该值可以是空字符串、All 或 Disabled。空字符串等同于指定 All。

3 可选：指定连接检查源 pod 的选择器。

4 可选：指定连接检查目标 pod 的选择器。

13.3. 配置 POD 连接检查放置

作为集群管理员，您可以通过修改名为 cluster 的 network.config.openshift.io 对象来配置运行连接 pod 的节点。

先决条件

- 安装 OpenShift CLI（oc）。

流程

1. 要编辑连接检查配置，请输入以下命令：

```
$ oc edit network.config.openshift.io cluster
```

2. 在文本编辑器中，更新 networkDiagnostics 小节，以指定您要用于源和目标 pod 的节点选择器。

3. 要提交您的更改，请保存更改并退出文本编辑器。

验证

要验证源和目标 pod 是否在预期节点上运行，请输入以下命令：

```
$ oc get pods -n openshift-network-diagnostics -o wide
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE    IP           NODE
NOMINATED NODE   READINESS GATES
network-check-source-84c69dbd6b-p8f7n   1/1     Running   0          9h     10.131.0.8   ip-10-0-40-
197.us-east-2.compute.internal   <none>           <none>
network-check-target-46pct              1/1     Running   0          9h     10.131.0.6   ip-10-0-40-197.us-
east-2.compute.internal   <none>           <none>
network-check-target-8kwgf              1/1     Running   0          9h     10.128.2.4   ip-10-0-95-74.us-
east-2.compute.internal   <none>           <none>
```

```
network-check-target-jc6n7          1/1     Running   0          9h     10.129.2.4   ip-10-0-21-151.us-
east-2.compute.internal   <none>           <none>
network-check-target-lvwnn          1/1     Running   0          9h     10.128.0.7   ip-10-0-17-129.us-
east-2.compute.internal   <none>           <none>
network-check-target-nslvj          1/1     Running   0          9h     10.130.0.7   ip-10-0-89-148.us-
east-2.compute.internal   <none>           <none>
network-check-target-z2sfx          1/1     Running   0          9h     10.129.0.4   ip-10-0-60-253.us-
east-2.compute.internal   <none>           <none>
```

13.4. PODNETWORKCONNECTIVITYCHECK 对象字段

PodNetworkConnectivityCheck 对象字段在下表中描述。

表 13.1. PodNetworkConnectivityCheck 对象字段

字段	类型	描述
metadata.name	**字符串**	对象的名称，其格式如下：**<source>-to-<target>**。**<target>** 描述的目的地包括以下字符串之一： • **load-balancer-api-external** • **load-balancer-api-internal** • **kubernetes-apiserver-endpoint** • **kubernetes-apiserver-service-cluster** • **network-check-target** • **openshift-apiserver-endpoint** • **openshift-apiserver-service-cluster**
metadata.namespace	**字符串**	与对象关联的命名空间。此值始终为 **openshift-network-diagnostics**。
spec.sourcePod	**字符串**	连接检查来源于的 pod 的名称，如 **network-check-source-596b4c6566-rgh92**。
spec.targetEndpoint	**字符串**	连接检查的目标，如 **api.devcluster.example.com:6443**。
spec.tlsClientCert	**对象**	要使用的 TLS 证书配置。
spec.tlsClientCert.name	**字符串**	使用的 TLS 证书的名称（若有）。默认值为空字符串。
status	**对象**	代表连接测试条件和最近连接发生和失败的日志的对象。
status.conditions	**数组**	连接检查以及任何之前的状态的最新状态。

字段	类型	描述
status.failures	**数组**	连接测试日志不会失败。
status.outages	**数组**	涵盖任何中断的时间连接测试日志。
status.successes	**数组**	成功尝试的连接测试日志。

下表描述了 status.conditions 阵列中对象的字段：

表 13.2. status.conditions

字段	类型	描述
lastTransitionTime	**字符串**	连接条件从一个状态转换到另一个状态的时间。
message	**字符串**	有关最后一次转换的详情（人类可读的格式）。
reason	**字符串**	有关最后一次转换的详情（机器可读的格式）。
status	**字符串**	条件的状态。
type	**字符串**	条件的类型。

下表描述了 status.conditions 阵列中对象的字段：

表 13.3. status.outages

字段	类型	描述
end	**字符串**	连接失败时的时间戳。
endLogs	**数组**	连接日志条目，包括与成功关闭相关的日志条目。
message	**字符串**	以人类可读格式显示停机详情概述。
开始	**字符串**	第一次检测到连接失败时的时间戳。
startLogs	**数组**	连接日志条目，包括原始失败。

连接日志字段

下表中描述了连接日志条目的字段。该对象用于以下字段：

- **status.failures[]**
- **status.successes[]**
- **status.outages[].startLogs[]**

- status.outages[].endLogs[]

表 13.4. 连接日志对象

字段	类型	描述
latency	字符串	记录操作的持续时间。
message	字符串	以人类可读格式提供的状态信息。
reason	字符串	以可读格式提供状态的原因。这个值是 TCPConnect、TCPConnectError、DNSResolve、DNSError 之一。
success	布尔值	指明日志条目是否成功或失败。
time	字符串	连接检查的开始时间。

13.5. 验证端点的网络连接

作为集群管理员，您可以验证端点的连接，如 API 服务器、负载均衡器、服务或 pod，并验证是否启用了网络诊断。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 角色的用户访问集群。

流程

1. 要确认启用了网络诊断，请输入以下命令：

```
$ oc get network.config.openshift.io cluster -o yaml
```

输出示例

```
# ...
status:
# ...
  conditions:
  - lastTransitionTime: "2024-05-27T08:28:39Z"
    message: ""
    reason: AsExpected
    status: "True"
    type: NetworkDiagnosticsAvailable
```

2. 要列出当前的 PodNetworkConnectivityCheck 对象，请输入以下命令：

```
$ oc get podnetworkconnectivitycheck -n openshift-network-diagnostics
```

输出示例

```
NAME                                                                                                                                AGE
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1   73m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-service-cluster                               75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-default-service-cluster                                 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-external                                         75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-internal                                         75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-0            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-1            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-2            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-c-n8mbf      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-service-cluster                               75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0    75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1    75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-service-cluster                                75m
```

3. 查看连接测试日志：

   a. 在上一命令的输出中，标识您要查看连接日志的端点。

   b. 要查看对象，请输入以下命令：

```
$ oc get podnetworkconnectivitycheck <name> \
  -n openshift-network-diagnostics -o yaml
```

   这里的 <name> 指定 PodNetworkConnectivityCheck 对象的名称。

   输出示例

```
apiVersion: controlplane.operator.openshift.io/v1alpha1
kind: PodNetworkConnectivityCheck
```

```
metadata:
  name: network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-
kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0
  namespace: openshift-network-diagnostics
  ...
spec:
  sourcePod: network-check-source-7c88f6d9f-hmg2f
  targetEndpoint: 10.0.0.4:6443
  tlsClientCert:
    name: ""
status:
  conditions:
  - lastTransitionTime: "2021-01-13T20:11:34Z"
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnectSuccess
    status: "True"
    type: Reachable
  failures:
  - latency: 2.241775ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:10:34Z"
  - latency: 2.582129ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:09:34Z"
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
  outages:
  - end: "2021-01-13T20:11:34Z"
    endLogs:
    - latency: 2.032018ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        tcp connection to 10.0.0.4:6443 succeeded'
      reason: TCPConnect
      success: true
      time: "2021-01-13T20:11:34Z"
    - latency: 2.241775ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
```

```
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:10:34Z"
  - latency: 2.582129ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:09:34Z"
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
  message: Connectivity restored after 2m59.999789186s
  start: "2021-01-13T20:08:34Z"
  startLogs:
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
 successes:
 - latency: 2.845865ms
   message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
     connection to 10.0.0.4:6443 succeeded'
   reason: TCPConnect
   success: true
   time: "2021-01-13T21:14:34Z"
 - latency: 2.926345ms
   message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
     connection to 10.0.0.4:6443 succeeded'
   reason: TCPConnect
   success: true
   time: "2021-01-13T21:13:34Z"
 - latency: 2.895796ms
   message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
     connection to 10.0.0.4:6443 succeeded'
   reason: TCPConnect
   success: true
   time: "2021-01-13T21:12:34Z"
 - latency: 2.696844ms
   message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
     connection to 10.0.0.4:6443 succeeded'
   reason: TCPConnect
   success: true
```

```
    time: "2021-01-13T21:11:34Z"
  - latency: 1.502064ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:10:34Z"
  - latency: 1.388857ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:09:34Z"
  - latency: 1.906383ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:08:34Z"
  - latency: 2.089073ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:07:34Z"
  - latency: 2.156994ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:06:34Z"
  - latency: 1.777043ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:05:34Z"
```

第 14 章 更改集群网络的 MTU

作为集群管理员，您可以在集群安装后更改集群网络的 MTU。这一更改具有破坏性，因为必须重启集群节点才能完成 MTU 更改。您只能为使用 OVN-Kubernetes 或 OpenShift SDN 网络插件的集群更改 MTU。

14.1. 关于集群 MTU

在安装集群网络的最大传输单元(MTU)期间，会根据集群中节点的主网络接口的 MTU 自动检测到。您通常不需要覆盖检测到的 MTU。

您可能希望因为以下原因更改集群网络的 MTU：

- 集群安装过程中检测到的 MTU 不正确。
- 集群基础架构现在需要不同的 MTU，如添加需要不同 MTU 的节点来获得最佳性能

只有 OVN-Kubernetes 集群网络插件支持更改 MTU 值。

14.1.1. 服务中断注意事项

当您为集群启动 MTU 更改时，以下效果可能会影响服务可用性：

- 至少需要两个滚动重启才能完成迁移到新的 MTU。在此过程中，一些节点在重启时不可用。
- 部署到集群的特定应用程序带有较短的超时间隔，超过绝对 TCP 超时间隔可能会在 MTU 更改过程中造成中断。

14.1.2. MTU 值选择

在规划 MTU 迁移时，需要考虑两个相关但不同的 MTU 值。

- Hardware MTU ：此 MTU 值根据您的网络基础架构的具体设置。
- Cluster network MTU ：此 MTU 值始终小于您的硬件 MTU，以考虑集群网络覆盖开销。具体开销由您的网络插件决定。对于 OVN-Kubernetes，开销为 100 字节。

如果您的集群为不同的节点需要不同的 MTU 值，则必须从集群中任何节点使用的最低 MTU 值中减去网络插件的开销值。例如，如果集群中的某些节点的 MTU 为 9001，而某些节点的 MTU 为 1500，则必须将此值设置为 1400。

重要

为了避免选择节点无法接受的 MTU 值，请使用 ip -d link 命令验证网络接口接受的最大 MTU 值 (maxmtu)。

14.1.3. 迁移过程如何工作

下表对迁移过程进行了概述，它分为操作中的用户发起的步骤，以及在响应过程中迁移过程要执行的操作。

表 14.1. 集群 MTU 的实时迁移

用户发起的步骤	OpenShift Container Platform 活动
在 Cluster Network Operator 配置中设置以下值： • **spec.migration.mtu.machine.to** • **spec.migration.mtu.network.from** • **spec.migration.mtu.network.to**	**Cluster Network Operator(CNO)**：确认每个字段都设置为有效的值。 • 如果硬件的 MTU 没有改变，则 **mtu.machine.to** 必须设置为新硬件 MTU 或当前的硬件 MTU。这个值是临时的，被用作迁移过程的一部分。如果您指定了与现有硬件 MTU 值不同的硬件 MTU，您必须手动将 MTU 配置为持久，如机器配置、DHCP 设置或 Linux 内核命令行。 • **mtu.network.from** 字段必须等于 **network.status.clusterNetworkMTU** 字段，这是集群网络的当前 MTU。 • **mtu.network.to** 字段必须设置为目标集群网络 MTU，且必须小于硬件 MTU，以允许网络插件的覆盖开销。对于 OVN-Kubernetes，开销为 **100** 字节。 如果提供的值有效，CNO 会生成一个新的临时配置，它将集群网络集的 MTU 设置为 **mtu.network.to** 字段的值。 **Machine Config Operator(MCO)**：执行集群中每个节点的滚动重启。
重新配置集群中节点的主网络接口 MTU。您可以使用各种方法完成此操作，包括： • 使用 MTU 更改部署新的 NetworkManager 连接配置集 • 通过 DHCP 服务器设置更改 MTU • 通过引导参数更改 MTU	N/A
在网络插件的 CNO 配置中设置 **mtu** 值，并将 **spec.migration** 设置为 **null**。	**Machine Config Operator(MCO)**：使用新的 MTU 配置执行集群中每个节点的滚动重启。

14.2. 更改集群网络 MTU

作为集群管理员，您可以增加或减少集群的最大传输单元 (MTU)。

重要

当 MTU 更新推出时，集群中的迁移具有破坏性且节点可能会临时不可用。

以下流程解释了如何使用机器配置、动态主机配置协议 (DHCP) 或 ISO 镜像更改集群网络 MTU。如果使用 DHCP 或 ISO 方法，则必须在安装集群后保留的配置工件来完成此流程。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 权限的账户访问集群。
- 已为集群识别目标 MTU。OVN-Kubernetes 网络插件的 MTU 必须设置为比集群中的最低硬件 MTU 值小 100。

流程

1. 要获得集群网络的当前 MTU，请输入以下命令：

```
$ oc describe network.config cluster
```

输出示例

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. 为硬件 MTU 准备配置：
   - 如果您的硬件 MTU 通过 DHCP 指定，请使用以下 dnsmasq 配置更新 DHCP 配置：

```
dhcp-option-force=26,<mtu>
```

其中：

<mtu>
指定要公告的 DHCP 服务器的硬件 MTU。

   - 如果使用 PXE 的内核命令行指定硬件 MTU，请相应地更新该配置。
   - 如果在 NetworkManager 连接配置中指定了硬件 MTU，请完成以下步骤。如果没有使用 DHCP、内核命令行或某种其他方法显式指定网络配置，则此方法是 OpenShift Container Platform 的默认方法。集群节点必须全部使用相同的底层网络配置，才能使以下过程未经修改地工作。
     i. 查找主网络接口：
        - 如果使用 OpenShift SDN 网络插件，请输入以下命令：

```
$ oc debug node/<node_name> -- chroot /host ip route list match 0.0.0.0/0 |
awk '{print $5 }'
```

其中：

<node_name>
指定集群中的节点的名称。

- 如果使用 OVN-Kubernetes 网络插件，请输入以下命令：

```
$ oc debug node/<node_name> -- chroot /host nmcli -g
connection.interface-name c show ovs-if-phys0
```

其中：

<node_name>
指定集群中的节点的名称。

ii. 在 <interface>-mtu.conf 文件中创建以下 NetworkManager 配置：

NetworkManager 连接配置示例

```
[connection-<interface>-mtu]
match-device=interface-name:<interface>
ethernet.mtu=<mtu>
```

其中：

<mtu>
指定新的硬件 MTU 值。

<interface>
指定主网络接口名称。

iii. 创建两个 MachineConfig 对象，一个用于 control plane 节点，另一个用于集群中的 worker 节点：

A. 在 control-plane-interface.bu 文件中创建以下 Butane 配置：

```
variant: openshift
version: 4.16.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf (1)
      contents:
        local: <interface>-mtu.conf (2)
      mode: 0600
```

(1) 指定主网络接口的 NetworkManager 连接名称。

(2) 指定上一步中更新的 NetworkManager 配置文件的本地文件名。

B. 在 worker-interface.bu 文件中创建以下 Butane 配置：

```
variant: openshift
version: 4.16.0
metadata:
  name: 01-worker-interface
```

```
    labels:
      machineconfiguration.openshift.io/role: worker
  storage:
    files:
      - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf (1)
        contents:
          local: <interface>-mtu.conf (2)
        mode: 0600
```

(1) 指定主网络接口的 NetworkManager 连接名称。

(2) 指定上一步中更新的 NetworkManager 配置文件的本地文件名。

C. 运行以下命令，从 Butane 配置创建 MachineConfig 对象：

```
$ for manifest in control-plane-interface worker-interface; do
    butane --files-dir . $manifest.bu > $manifest.yaml
  done
```

> **警告**
>
> 在此流程的稍后明确指示之前，不要应用这些机器配置。应用这些机器配置现在会导致集群的稳定性丢失。

3. 要开始 MTU 迁移，请输入以下命令指定迁移配置。Machine Config Operator 在集群中执行节点的滚动重启，以准备 MTU 更改。

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to":
<overlay_to> } , "machine": { "to" : <machine_to> } } } } }'
```

其中：

<overlay_from>
: 指定当前的集群网络 MTU 值。

<overlay_to>
: 指定集群网络的目标 MTU。这个值相对于 <machine_to> 的值设置。对于 OVN-Kubernetes，这个值必须比 <machine_to> 的值小 100。

<machine_to>
: 指定底层主机网络上的主网络接口的 MTU。

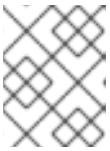
增加集群 MTU 的示例

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": {
"to" : 9100} } } } }'
```

4. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态：UPDATED=true、UPDATING=false、DEGRADED=false。



注意

默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

5. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

```
$ oc describe node | egrep "hostname|machineconfig"
```

   输出示例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

   b. 验证以下语句是否正确：

      - machineconfiguration.openshift.io/state 字段的值为 Done。
      - machineconfiguration.openshift.io/currentConfig 字段的值等于 machineconfiguration.openshift.io/desiredConfig 字段的值。

   c. 要确认机器配置正确，请输入以下命令：

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

   这里的 <config_name> 是 machineconfiguration.openshift.io/currentConfig 字段中机器配置的名称。

   机器配置必须包括以下对 systemd 配置的更新：

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

6. 更新底层网络接口 MTU 值：

   - 如果您要使用 NetworkManager 连接配置指定新 MTU，请输入以下命令。MachineConfig Operator 会自动执行集群中节点的滚动重启。

```
$ for manifest in control-plane-interface worker-interface; do
    oc create -f $manifest.yaml
  done
```

- 如果您要使用 DHCP 服务器选项或内核命令行和 PXE 指定新 MTU，请对基础架构进行必要的更改。

7. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态：UPDATED=true、UPDATING=false、DEGRADED=false。

注意

默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

8. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

```
$ oc describe node | egrep "hostname|machineconfig"
```

   输出示例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

   验证以下语句是否正确：

   - machineconfiguration.openshift.io/state 字段的值为 Done。
   - machineconfiguration.openshift.io/currentConfig 字段的值等于 machineconfiguration.openshift.io/desiredConfig 字段的值。

   b. 要确认机器配置正确，请输入以下命令：

```
$ oc get machineconfig <config_name> -o yaml | grep path:
```

   这里的 <config_name> 是 machineconfiguration.openshift.io/currentConfig 字段中机器配置的名称。

   如果机器配置被成功部署，则前面的输出会包含 /etc/NetworkManager/conf.d/99-<interface>-mtu.conf 文件路径和 ExecStart=/usr/local/bin/mtu-migration.sh 行。

9. 要完成 MTU 迁移，请为 OVN-Kubernetes 网络插件输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu":
<mtu> }}}}'
```

其中：

<mtu>
指定您使用 <overlay_to> 指定的新集群网络 MTU。

10. 最终调整 MTU 迁移后，每个机器配置池节点都会逐个重启一个。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态：UPDATED=true、UPDATING=false、DEGRADED=false。

验证

1. 要获得集群网络的当前 MTU，请输入以下命令：

```
$ oc describe network.config cluster
```

2. 获取节点的主网络接口的当前 MTU：
   a. 要列出集群中的节点，请输入以下命令：

```
$ oc get nodes
```

   b. 要获取节点上主网络接口的当前 MTU 设置，请输入以下命令：

```
$ oc debug node/<node> -- chroot /host ip address show <interface>
```

其中：

<node>
指定上一步中的输出节点。
<interface>
指定节点的主网络接口名称。

输出示例

```
ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8051
```

14.3. 其他资源

- 使用高级网络选项进行 PXE 和 ISO 安装
- 使用密钥文件格式手动创建 NetworkManager 配置集
- 使用 nmcli 配置动态以太网连接

第 15 章 配置节点端口服务范围

作为集群管理员，您可以扩展可用的节点端口范围。如果您的集群使用大量节点端口，可能需要增加可用端口的数量。

默认端口范围为 30000-32767。您永远不会缩小端口范围，即使您首先将其扩展超过默认范围。

15.1. 先决条件

- 集群基础架构必须允许访问您在扩展范围内指定的端口。例如，如果您将节点端口范围扩展到 30000-32900，防火墙或数据包过滤配置必须允许 32768-32900 端口范围。

15.2. 扩展节点端口范围

您可以扩展集群的节点端口范围。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 要扩展节点端口范围，请输入以下命令。将 <port> 替换为新范围内的最大端口号码。

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
  '{
    "spec":
      { "serviceNodePortRange": "30000-<port>" }
  }'
```

提示

您还可以应用以下 YAML 来更新节点端口范围：

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

输出示例

```
network.config.openshift.io/cluster patched
```

2. 要确认配置是活跃的，请输入以下命令。应用更新可能需要几分钟。

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo '"service-node-port-range":["[[:digit:]]+-[[:digit:]]+"]'
```

-

输出示例

```
"service-node-port-range":["30000-33000"]
```

15.3. 其他资源

- 使用 NodePort 配置集群入口流量
- Network [config.openshift.io/v1]
- 服务 [core/v1]

第 16 章 配置集群网络范围

作为集群管理员，您可以在集群安装后扩展集群网络范围。如果额外的节点需要更多 IP 地址，您可能需要扩展集群网络范围。

例如，如果您部署了集群，并将 10.128.0.0/19 指定为集群网络范围，主机前缀为 23，则限制为 16 个节点。您可以通过将集群中的 CIDR 掩码更改为 /14 来扩展到 510 个节点。

在扩展集群网络地址范围时，您的集群必须使用 OVN-Kubernetes 网络插件。不支持其他网络插件。

修改集群网络 IP 地址范围时会有以下限制：

- 指定的 CIDR 掩码大小必须总是小于当前配置的 CIDR 掩码大小，因为您只能向已安装的集群添加更多节点来增加 IP 空间
- 无法修改主机前缀
- 使用覆盖默认网关配置的 Pod 必须在集群网络扩展后重新创建

16.1. 扩展集群网络 IP 地址范围

您可以扩展集群网络的 IP 地址范围。由于这个更改需要在集群中推出新的 Operator 配置，所以最多可能需要 30 分钟才能生效。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 确保集群使用 OVN-Kubernetes 网络插件。

流程

1. 要获取集群的集群网络范围和主机前缀，请输入以下命令：

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

输出示例

```
[{"cidr":"10.217.0.0/22","hostPrefix":23}]
```

2. 要扩展集群网络 IP 地址范围，请输入以下命令。使用上一命令输出返回的 CIDR IP 地址范围和主机前缀。

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
    "spec":{
      "clusterNetwork": [ {"cidr":"<network>/<cidr>","hostPrefix":<prefix>} ],
      "networkType": "OVNKubernetes"
    }
  }'
```

其中：

<network>
指定您在上一步中获取的 cidr 字段的网络部分。您无法更改这个值。

<cidr>
指定网络前缀长度。例如，14。将此值更改为比上一步中的输出值小的值，以扩展集群网络范围。

<prefix>
指定集群的当前主机前缀。这个值必须与您在上一步中获取的 hostPrefix 字段的值相同。

示例命令

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
    "spec":{
      "clusterNetwork": [ {"cidr":"10.217.0.0/14","hostPrefix": 23} ],
      "networkType": "OVNKubernetes"
    }
  }'
```

输出示例

```
network.config.openshift.io/cluster patched
```

3. 要确认配置是活跃的，请输入以下命令。可能需要 30 分钟才能使此更改生效。

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

输出示例

```
[{"cidr":"10.217.0.0/14","hostPrefix":23}]
```

16.2. 其他资源

- Red Hat OpenShift Network Calculator
- 关于 OVN-Kubernetes 网络插件

第 17 章 配置 IP 故障转移

本节论述了为 OpenShift Container Platform 集群上的 pod 和服务配置 IP 故障转移。

IP 故障转移使用 Keepalived 在一组主机上托管一组外部访问的虚拟 IP (VIP) 地址。每个 VIP 地址仅由单个主机提供服务。Keepalived 使用虚拟路由器冗余协议（VRRP）决定在主机集合中使用哪个主机提供 VIP 服务。如果主机不可用，或者 Keepalived 正在监视的服务没有响应，则 VIP 会切换到主机集中的另外一个主机。这意味着只要主机可用，便始终可以提供 VIP 服务。

集合中的每个 VIP 都由从集合中选择的节点提供服务。如果单个节点可用，则会提供 VIP。无法将 VIP 显式分发到节点上，因此可能存在没有 VIP 的节点和其他具有多个 VIP 的节点。如果只有一个节点，则所有 VIP 都在其中。

管理员必须确保所有 VIP 地址都满足以下要求：

- 可在集群外部配置的主机上访问。
- 不用于集群中的任何其他目的。

每个节点上的 keepalived 确定所需服务是否在运行。如果是，则支持 VIP，Keepalived 参与协商来确定哪个节点服务 VIP。对于要参与的节点，服务必须侦听 VIP 上的观察端口，或者必须禁用检查。

注意

集合中的每个 VIP 都可以由不同的节点提供。

IP 故障转移会监控每个 VIP 上的端口，以确定该端口能否在节点上访问。如果端口无法访问，则不会向节点分配 VIP。如果端口设为 0，则会禁止此检查。检查脚本执行所需的测试。

当运行 Keepalived 的节点通过检查脚本时，该节点上的 VIP 可以根据其优先级和当前 master 的优先级以及抢占策略决定进入 master 状态。

集群管理员可以通过 OPENSHIFT_HA_NOTIFY_SCRIPT 变量提供一个脚本，每当节点上的 VIP 的状态发生变化时会调用此脚本。keepalived 在为 VIP 提供服务时为 master 状态；当另一个节点提供 VIP 服务时，状态为 backup；当检查脚本失败时，状态为 fault。每当状态更改时，notify 脚本都会被调用，并显示新的状态。

您可以在 OpenShift Container Platform 上创建 IP 故障转移部署配置。IP 故障转移部署配置指定 VIP 地址的集合，以及服务它们的一组节点。一个集群可以具有多个 IP 故障转移部署配置，各自管理自己的一组唯一的 VIP 地址。IP 故障转移配置中的每个节点运行 IP 故障转移 pod，此 pod 运行 Keepalived。

使用 VIP 访问带有主机网络的 pod 时，应用程序 pod 在运行 IP 故障转移 pod 的所有节点上运行。这可让任何 IP 故障转移节点成为主节点，并在需要时为 VIP 服务。如果应用程序 pod 没有在所有具有 IP 故障转移功能的节点上运行，有些 IP 故障转移节点不会为 VIP 服务，或者某些应用 pod 都不会接收任何流量。对 IP 故障转移和应用容器集使用相同的选择器和复制数，以避免这种不匹配。

在使用 VIP 访问服务时，任何节点都可以位于节点的 IP 故障转移集中，因为无论应用容器集在哪里运行，该服务都可以在所有节点上访问。任何 IP 故障转移节点可以随时变成主节点。服务可以使用外部 IP 和服务端口，或者可以使用 NodePort。设置 NodePort 是一个特权操作。

在服务定义中使用外部 IP 时，VIP 被设置为外部 IP，IP 故障转移监控端口则设为服务端口。在使用节点端口时，该端口在集群的每个节点上打开，服务则从当前服务于 VIP 的任何节点对流量进行负载平衡。在这种情况下，IP 故障转移监控端口在服务定义中设置为 NodePort。

重要

即使一个服务 VIP 具有高可用性，但性能仍会受到影响。keepalived 确保每个 VIP 都由配置中的某个节点提供服务，即使其他节点没有，也可以在同一节点上出现多个 VIP。当 IP 故障转移在同一节点上放置多个 VIP 时，在一组 VIP 间进行外部负载平衡的策略可能会被破解。

当使用 ExternalIP 时，您可以将 IP 故障转移设置为与 ExternalIP 范围相同的 VIP 范围。您还可以禁用监控端口。在这种情况下，所有 VIP 都出现在集群中的同一节点上。任何用户都可以使用 ExternalIP 设置服务并使其高度可用。

重要

集群中最多有 254 个 VIP。

17.1. IP 故障转移环境变量

下表包含用于配置 IP 故障转移的变量。

表 17.1. IP 故障转移环境变量

变量名称	default	描述
OPENSHIFT_HA_MONITOR_PORT	**80**	IP 故障转移 pod 会尝试在每个虚拟 IP（VIP）上打开到此端口的 TCP 连接。如果建立连接，则服务将被视为正在运行。如果此端口设为 **0**，则测试会始终通过。
OPENSHIFT_HA_NETWORK_INTERFACE		IP 故障转移用于发送虚拟路由器冗余协议 (VRRP) 流量的接口名称。默认值为 **eth0**。
OPENSHIFT_HA_REPLICA_COUNT	**2**	要创建的副本数。这必须与 IP 故障转移部署配置中的 **spec.replicas** 值匹配。
OPENSHIFT_HA_VIRTUAL_IPS		要复制的 IP 地址范围列表。必须提供.例如，**1.2.3.4-6,1.2.3.9**。
OPENSHIFT_HA_VRRP_ID_OFFSET	**0**	用于设置虚拟路由器 ID 的偏移值。使用不同的偏移值可以在同一集群中存在多个 IP 故障转移配置。默认偏移值为 **0**，允许的范围是 **0** 到 **255**。
OPENSHIFT_HA_VIP_GROUPS		为 VRRP 创建的组数量。如果没有设置，则会为通过 **OPENSHIFT_HA_VIP_GROUPS** 变量指定的每个虚拟 IP 范围创建一个组。
OPENSHIFT_HA_IPTABLES_CHAIN	输入	iptables 链的名称，用于自动添加允许 VRRP 流量的 **iptables** 规则。如果没有设置值，则不会添加 **iptables** 规则。如果链不存在，则不会创建它。
OPENSHIFT_HA_CHECK_SCRIPT		定期运行的脚本的 pod 文件系统中的完整路径名称，以验证应用是否正在运行。

变量名称	default	描述
OPENSHIFT_HA_CHECK_INTERVAL	2	检查脚本运行的期间（以秒为单位）。
OPENSHIFT_HA_NOTIFY_SCRIPT		当状态发生变化时运行的脚本的 pod 文件系统的完整路径名称。
OPENSHIFT_HA_PREEMPTION	preempt_nodelay 300	处理新的具有更高优先级主机的策略。**nopreempt** 策略不会将 master 从较低优先级主机移到优先级更高的主机。

17.2. 在集群中配置 IP 故障切换

作为集群管理员，您可以在整个集群中或在其中的一部分节点（由标签选项器定义）中配置 IP 故障转移。您还可以在集群中配置多个 IP 故障转移部署配置，每个配置都独立于其他配置。

IP 故障转移部署配置确保故障转移 pod 在符合限制或使用的标签的每个节点上运行。

此 pod 运行 Keepalived，它可以监控端点，并在第一个节点无法访问服务或端点时使用 Virtual Router Redundancy Protocol（VRRP）从一个节点切换到另一个节点的虚拟 IP（VIP）。

对于生产环境，设置一个选择器（selector），用于选择至少两个节点，并设置与所选节点数量相等的副本。

先决条件

- 使用具有 cluster-admin 权限的用户登陆到集群。
- 已创建一个 pull secret。

流程

1. 创建 IP 故障转移服务帐户：

```
$ oc create sa ipfailover
```

2. 为 **hostNetwork** 更新安全性上下文约束（SCC）：

```
$ oc adm policy add-scc-to-user privileged -z ipfailover
```

```
$ oc adm policy add-scc-to-user hostnetwork -z ipfailover
```

3. 创建部署 YAML 文件来配置 IP 故障转移：

   IP 故障转移配置的部署 YAML 示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ipfailover-keepalived (1)
```

```
  labels:
    ipfailover: hello-openshift
spec:
  strategy:
    type: Recreate
  replicas: 2
  selector:
    matchLabels:
      ipfailover: hello-openshift
  template:
    metadata:
      labels:
        ipfailover: hello-openshift
    spec:
      serviceAccountName: ipfailover
      privileged: true
      hostNetwork: true
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
      - name: openshift-ipfailover
        image: quay.io/openshift/origin-keepalived-ipfailover
        ports:
        - containerPort: 63000
          hostPort: 63000
        imagePullPolicy: IfNotPresent
        securityContext:
          privileged: true
        volumeMounts:
        - name: lib-modules
          mountPath: /lib/modules
          readOnly: true
        - name: host-slash
          mountPath: /host
          readOnly: true
          mountPropagation: HostToContainer
        - name: etc-sysconfig
          mountPath: /etc/sysconfig
          readOnly: true
        - name: config-volume
          mountPath: /etc/keepalive
        env:
        - name: OPENSHIFT_HA_CONFIG_NAME
          value: "ipfailover"
        - name: OPENSHIFT_HA_VIRTUAL_IPS 2
          value: "1.1.1.1-2"
        - name: OPENSHIFT_HA_VIP_GROUPS 3
          value: "10"
        - name: OPENSHIFT_HA_NETWORK_INTERFACE 4
          value: "ens3" #The host interface to assign the VIPs
        - name: OPENSHIFT_HA_MONITOR_PORT 5
          value: "30060"
        - name: OPENSHIFT_HA_VRRP_ID_OFFSET 6
          value: "0"
        - name: OPENSHIFT_HA_REPLICA_COUNT 7
```

```
      value: "2" #Must match the number of replicas in the deployment
    - name: OPENSHIFT_HA_USE_UNICAST
      value: "false"
    #- name: OPENSHIFT_HA_UNICAST_PEERS
      #value: "10.0.148.40,10.0.160.234,10.0.199.110"
    - name: OPENSHIFT_HA_IPTABLES_CHAIN 8
      value: "INPUT"
    #- name: OPENSHIFT_HA_NOTIFY_SCRIPT 9
    #  value: /etc/keepalive/mynotifyscript.sh
    - name: OPENSHIFT_HA_CHECK_SCRIPT 10
      value: "/etc/keepalive/mycheckscript.sh"
    - name: OPENSHIFT_HA_PREEMPTION 11
      value: "preempt_delay 300"
    - name: OPENSHIFT_HA_CHECK_INTERVAL 12
      value: "2"
    livenessProbe:
      initialDelaySeconds: 10
      exec:
        command:
        - pgrep
        - keepalived
  volumes:
  - name: lib-modules
    hostPath:
      path: /lib/modules
  - name: host-slash
    hostPath:
      path: /
  - name: etc-sysconfig
    hostPath:
      path: /etc/sysconfig
  # config-volume contains the check script
  # created with `oc create configmap keepalived-checkscript --from-
file=mycheckscript.sh`
  - configMap:
      defaultMode: 0755
      name: keepalived-checkscript
    name: config-volume
  imagePullSecrets:
    - name: openshift-pull-secret 13
```

1. IP 故障转移部署的名称。
2. 要复制的 IP 地址范围列表。必须提供.例如，1.2.3.4-6,1.2.3.9。
3. 为 VRRP 创建的组数量。如果没有设置，则会为通过 OPENSHIFT_HA_VIP_GROUPS 变量指定的每个虚拟 IP 范围创建一个组。
4. IP 故障切换用于发送 VRRP 流量的接口名称。默认情况下使用 eth0。
5. IP 故障转移 pod 会尝试在每个 VIP 上打开到此端口的 TCP 连接。如果建立连接，则服务将被视为正在运行。如果此端口设为 0，则测试会始终通过。默认值为 80。
6. 用于设置虚拟路由器 ID 的偏移值。使用不同的偏移值可以在同一集群中存在多个 IP 故障转移配置。默认偏移值为 0，允许的范围是 0 到 255。

7 要创建的副本数。这必须与 IP 故障转移部署配置中的 spec.replicas 值匹配。默认值为 2。

8 iptables 链的名称，用于自动添加允许 VRRP 流量的iptables 规则。如果没有设置值，则不会添加 iptables 规则。如果链不存在，则不会创建链，Keepalived 在单播模式下运行。默认为 INPUT。

9 当状态发生变化时运行的脚本的 pod 文件系统的完整路径名称。

10 定期运行的脚本的 pod 文件系统中的完整路径名称，以验证应用是否正在运行。

11 处理新的具有更高优先级主机的策略。默认值为 preempt_delay 300，这会导致，在有一个较低优先级的 master 提供 VIP 时，Keepalived 实例在 5 分钟后会接管 VIP。

12 检查脚本运行的期间（以秒为单位）。默认值为 2。

13 在创建部署之前创建 pull secret，否则您将在创建部署时收到错误。

17.3. 配置检查和通知脚本

keepalived 通过定期运行可选用户提供的检查脚本来监控应用程序的健康状况。例如，该脚本可以通过发出请求并验证响应来测试 Web 服务器。作为集群管理员，您可以提供一个可选的 notify 脚本，该脚本会在状态发生变化时调用。

检查和通知在 IP 故障转移容器集中运行的脚本，并使用容器集文件系统，而不是主机文件系统。但是，IP 故障转移 pod 使主机文件系统在 /hosts 挂载路径下可用。在配置检查或通知脚本时，您必须提供脚本的完整路径。提供脚本的建议方法是使用 ConfigMap 对象。

检查和通知脚本的完整路径名称添加到 Keepalived 配置文件 _/etc/keepalived/keepalived.conf 中，该文件会在 Keepalived 每次启动时加载。可以使用 ConfigMap 对象将脚本添加到 pod，如以下方法所述。

检查脚本

不提供检查脚本时，将运行一个简单的默认脚本来测试 TCP 连接。当监控端口为 0 时，禁止此默认测试。

每个 IP 故障转移 pod 管理一个 Keepalived 守护进程，在运行 pod 的节点上管理一个或多个虚拟 IP (VIP) 地址。Keepalived 守护进程为该节点保留每个 VIP 的状态。特定节点上的特定 VIP 可能处于 master、backup 或 fault 状态。

如果检查脚本返回非零，节点会进入 backup 状态，并且它拥有的任何 VIP 被重新分配。

notify 脚本

keepalived 将以下三个参数传递给 notify 脚本：

- $1 - group 或 instance
- $2 - group 或 instance 的名称
- $3 - 新状态：master、backup 或 fault

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 创建所需脚本，并创建 ConfigMap 对象来容纳它。脚本没有输入参数，并且必须返回 0（OK）和 1（fail）。
   检查脚本，*mycheckscript.sh*：

```
#!/bin/bash
   # Whatever tests are needed
   # E.g., send request and verify response
exit 0
```

2. 创建 ConfigMap 对象：

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. 将脚本添加到容器集。挂载的 ConfigMap 对象的 defaultMode 必须能够使用 oc 命令或编辑部署配置来运行。值通常为 0755、493（十进制）：

```
$ oc set env deploy/ipfailover-keepalived \
   OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
```

```
$ oc set volume deploy/ipfailover-keepalived --add --overwrite \
   --name=config-volume \
   --mount-path=/etc/keepalive \
   --source='{"configMap": { "name": "mycustomcheck", "defaultMode": 493}}'
```

注意

oc set env 命令对空格敏感。= 符号的两侧不能有空格。

提示

您还可以编辑 ipfailover-keepalived 部署配置：

```
$ oc edit deploy ipfailover-keepalived
```

```
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_CHECK_SCRIPT (1)
          value: /etc/keepalive/mycheckscript.sh
...
        volumeMounts: (2)
        - mountPath: /etc/keepalive
          name: config-volume
      dnsPolicy: ClusterFirst
...
      volumes: (3)
      - configMap:
          defaultMode: 0755 (4)
          name: customrouter
        name: config-volume
...
```

1. 在 spec.container.env 字段中，添加 OPENSHIFT_HA_CHECK_SCRIPT 环境变量以指向挂载的脚本文件。
2. 添加 spec.container.volumeMounts 字段以创建挂载点。
3. 添加新的 spec.volumes 字段以提及配置映射。
4. 这将设置文件的运行权限。在重新读后，其显示为十进制 493。

保存更改并退出编辑器。这会重启 ipfailover-keepalived。

17.4. 配置 VRRP 抢占

当一个节点上的虚拟 IP（VIP）因为通过了检查脚本的检查而脱离 fault 状态时，如果其优先级低于当前处于 master 状态的节点上的 VIP，则节点上的 VIP 将进入backup 状态。nopreempt 策略不会将 master 从主机上的较低优先级 VIP 移到主机上的优先级更高的 VIP。当使用默认的 preempt_delay 300 时，Keepalived 会等待指定的 300 秒，并将 master 移到主机上的优先级更高的 VIP。

流程

- 要指定抢占，输入 oc edit deploy ipfailover-keepalived 以编辑路由器部署配置：

```
$ oc edit deploy ipfailover-keepalived
```

```
...
    spec:
      containers:
      - env:
        - name: OPENSHIFT_HA_PREEMPTION (1)
          value: preempt_delay 300
...
```

1. 设置 OPENSHIFT_HA_PREEMPTION 值：
   - preempt_delay 300：Keepalived 会等待指定的 300 秒，并将master 移到主机上的优先级更高的 VIP。这是默认值。
   - nopreempt：不会将 master 从主机上的较低优先级 VIP 移到主机上的优先级更高的 VIP。

17.5. 部署多个 IP 故障转移实例

每个 IP 转移 pod 由 IP 故障转移部署配置管理，每个节点 1 个 pod，以一个 Keepalived 守护进程运行。配置更多 IP 故障转移部署配置后，会创建更多 pod，更多的守护进程加入常见的虚拟路由器冗余协议（VRRP）协商。此协商由所有 Keepalived 守护进程完成，它决定了哪些节点服务是哪个虚拟 IP（VIP）。

Keepalived 内部为每个 VIP 分配一个唯一的 vrrp-id。协商使用这一组 vrrp-ids，在做出决策时，胜出的 vrrp-id 对应的 VIP 将在胜出的节点上服务。

因此，对于 IP 故障转移部署配置中定义的每个 VIP，IP 故障转移 pod 必须分配对应的 vrrp-id。这可以从 OPENSHIFT_HA_VRRP_ID_OFFSET 开始，并按顺序将 vrrp-ids 分配到 VIP 列表来实现。vrrp-ids 的值可在 1..255 之间。

当存在多个 IP 故障转移部署配置时，您必须指定 OPENSHIFT_HA_VRRP_ID_OFFSET，以便在部署配置中增加 VIP 的数量，并且没有 vrrp-id 范围重叠。

17.6. 为超过 254 地址配置 IP 故障转移

IP 故障转移管理有 254 个组虚拟 IP（VIP）地址的限制。默认情况下，OpenShift Container Platform 会为每个组分配一个 IP 地址。您可以使用 OPENSHIFT_HA_VIP_GROUPS 变量进行更改，使得每个组中有多个 IP 地址，并在配置 IP 故障转移时定义每个虚拟路由器冗余协议（VRRP）实例可用的 VIP 组数量。

在 VRRP 故障转移事件中，对 VIP 进行分组会为每个 VRRP 创建更广泛的 VIP 分配范围，并在集群中的所有主机都能够从本地访问服务时很有用。例如，当服务通过 ExternalIP 公开时。

注意

使用故障转移的一个规则是，请勿将路由等服务限制到一个特定的主机。相反，服务应复制到每一主机上，以便在 IP 故障转移时，不必在新主机上重新创建服务。

注意

如果使用 OpenShift Container Platform 健康检查，IP 故障转移和组的性质意味着不会检查组中的所有实例。因此，必须使用 Kubernetes 健康检查来确保服务处于活动状态。

先决条件

- 使用具有 cluster-admin 权限的用户登陆到集群。

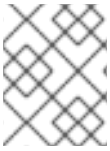
流程

- 要更改分配给每个组的 IP 地址数量，请更改 OPENSHIFT_HA_VIP_GROUPS 变量的值，例如：

IP 故障转换配置的 Deployment YAML 示例

```
...
  spec:
    env:
    - name: OPENSHIFT_HA_VIP_GROUPS (1)
      value: "3"
...
```

(1) 如果在有七个 VIP 的环境中将 OPENSHIFT_HA_VIP_GROUPS 设置为 3，它会创建三个组，将三个 VIP 分配到第一个组，为剩余的两个组各分配两个 VIP。



注意

如果 OPENSHIFT_HA_VIP_GROUPS 设置的组数量少于设置为故障的 IP 地址数量，则组包含多个 IP 地址，且所有地址都作为一个单元移动。

17.7. EXTERNALIP 的高可用性

在非云集群中，可以组合使用 IP 故障切换和 ExternalIP 到服务。对于使用 ExternalIP 创建服务的用户，结果是高可用性服务。

方法是指定集群网络配置的 spec.ExternalIP.autoAssignCIDRs 范围，然后在创建 IP 故障转移配置时使用相同的范围。

因为 IP 故障转移最多可支持整个集群的 255 个 VIP，所以 spec.ExternalIP.autoAssignCIDRs 必须为 /24 或更小。

17.8. 删除 IP 故障切换

在初始配置 IP 故障切换时，集群中的 worker 节点会使用 iptables 规则修改，该规则明确允许 Keepalived 在 224.0.0.18 上多播数据包。由于对节点的更改，移除 IP 故障切换需要运行一个作业来删除 iptables 规则并删除 Keepalived 使用的虚拟 IP 地址。

流程

1. 可选：识别并删除存储为配置映射的任何检查和通知脚本：
   a. 确定任何用于 IP 故障切换的 pod 是否使用配置映射作为卷：

```
$ oc get pod -l ipfailover \
  -o jsonpath="\
{range .items[?(@.spec.volumes[*].configMap)]}
{'Namespace: '}{.metadata.namespace}
{'Pod:       '}{.metadata.name}
{'Volumes that use config maps:'}
{range .spec.volumes[?(@.configMap)]}  {'volume:    '}{.name}
  {'configMap: '}{.configMap.name}{'\n'}{end}
{end}"
```

输出示例

```
Namespace: default
Pod:       keepalived-worker-59df45db9c-2x9mn
Volumes that use config maps:
  volume:    config-volume
  configMap: mycustomcheck
```

b. **如果上一步提供了用作卷的配置映射的名称，请删除配置映射：**

```
$ oc delete configmap <configmap_name>
```

2. 为 IP 故障切换识别现有部署：

```
$ oc get deployment -l ipfailover
```

输出示例

```
NAMESPACE   NAME       READY   UP-TO-DATE   AVAILABLE   AGE
default     ipfailover   2/2     2            2           105d
```

3. **删除部署：**

```
$ oc delete deployment <ipfailover_deployment_name>
```

4. **删除 ipfailover 服务帐户：**

```
$ oc delete sa ipfailover
```

5. 运行一个作业，该作业会删除最初配置 IP 故障切换时添加的 IP 表规则：

a. 创建一个文件，如 remove-ipfailover-job.yaml，其内容类似以下示例：

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: remove-ipfailover-
  labels:
    app: remove-ipfailover
spec:
  template:
    metadata:
      name: remove-ipfailover
    spec:
      containers:
      - name: remove-ipfailover
        image: quay.io/openshift/origin-keepalived-ipfailover:4.16
        command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
      nodeSelector: 1
        kubernetes.io/hostname: <host_name> 2
      restartPolicy: Never
```

1 nodeSelector 可能与旧 IP 故障转移部署中使用的选择器相同。

2 为集群中配置 IP 故障切换的每个节点运行作业，并每次替换主机名。

b. 运行作业：

```
$ oc create -f remove-ipfailover-job.yaml
```

输出示例

```
job.batch/remove-ipfailover-2h8dm created
```

验证

- 确认作业删除了 IP 故障切换的初始配置。

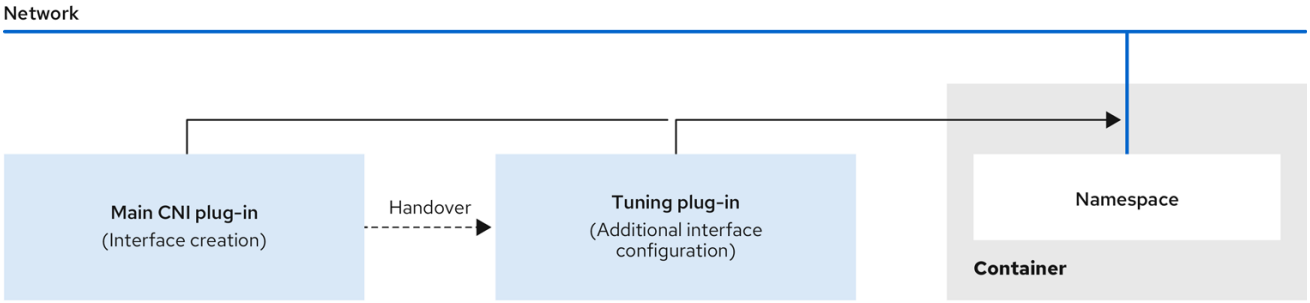
```
$ oc logs job/remove-ipfailover-2h8dm
```

输出示例

```
remove-failover.sh: OpenShift IP Failover service terminating.
  - Removing ip_vs module ...
  - Cleaning up ...
  - Releasing VIPs  (interface eth0) ...
```

第 18 章 使用调优插件配置系统控制和接口属性

在 Linux 中，管理员可通过 sysctl 在运行时修改内核参数。您可以使用调优 Container Network Interface(CNI)元插件修改接口级网络 sysctl。tuning CNI meta 插件在一个链中运行，主 CNI 插件如下所示。



主 CNI 插件分配接口，并将此接口在运行时传递给 tuning CNI meta 插件。您可以使用 tuning CNI meta 插件更改网络命名空间中的一些 sysctl 和几个接口属性，如 promiscuous 模式、all-multicast 模式、MTU 和 MAC 地址。

18.1. 使用 TUNING CNI 配置系统控制

以下流程将调整 CNI 配置为更改接口级网络 **net.ipv4.conf.IFNAME.accept_redirects** sysctl。这个示例启用接受和发送 ICMP 重定向的数据包。在 tuning CNI meta 插件配置中，接口名称由 **IFNAME** 令牌表示，并替换为运行时接口的实际名称。

流程

1. 使用以下内容创建网络附加定义，如 **tuning-example.yaml** ：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
  namespace: default 2
spec:
  config: '{
    "cniVersion": "0.4.0", 3
    "name": "<name>", 4
    "plugins": [{
      "type": "<main_CNI_plugin>" 5
    },
    {
     "type": "tuning", 6
     "sysctl": {
         "net.ipv4.conf.IFNAME.accept_redirects": "1" 7
     }
    }
   ]
}
```

1. 指定要创建的额外网络附加的名称。名称在指定的命名空间中必须是唯一的。
2. 指定与对象关联的命名空间。
3. 指定 CNI 规格版本。
4. 指定配置的名称。建议您将配置名称与网络附加定义的 name 值匹配。
5. 指定要配置的主 CNI 插件的名称。
6. 指定 CNI meta 插件的名称。
7. 指定要设置的 sysctl。接口名称由 IFNAME 令牌表示，并替换为运行时接口的实际名称。

下面显示了一个 YAML 文件示例：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
    },
    {
      "type": "tuning",
      "sysctl": {
        "net.ipv4.conf.IFNAME.accept_redirects": "1"
      }
    }
  ]
}'
```

2. 运行以下命令来应用 YAML：

```
$ oc apply -f tuning-example.yaml
```

输出示例

```
networkattachmentdefinition.k8.cni.cncf.io/tuningnad created
```

3. 使用类似以下示例的网络附加定义，创建示例 pod.yaml ：

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad 1
```

```
spec:
 containers:
 - name: podexample
   image: centos
   command: ["/bin/bash", "-c", "sleep INF"]
   securityContext:
     runAsUser: 2000 2
     runAsGroup: 3000 3
     allowPrivilegeEscalation: false 4
     capabilities: 5
       drop: ["ALL"]
 securityContext:
   runAsNonRoot: true 6
   seccompProfile: 7
     type: RuntimeDefault
```

1. 指定配置的 NetworkAttachmentDefinition 的名称。
2. runAsUser 控制使用哪个用户 ID 运行容器。
3. runAsGroup 控制容器使用哪个主要组 ID。
4. allowPrivilegeEscalation 决定 pod 是否请求允许特权升级。如果未指定，则默认为 true。这个布尔值直接控制在容器进程中是否设置了 no_new_privs 标志。
5. capabilities 允许特权操作，而不提供完整的 root 访问权限。此策略可确保从 pod 中丢弃了所有功能。
6. runAsNonRoot: true 要求容器使用 0 以外的任何 UID 运行。
7. RuntimeDefault 为 pod 或容器工作负载启用默认的 seccomp 配置集。

4. 运行以下命令来应用 yaml:

```
$ oc apply -f examplepod.yaml
```

5. 运行以下命令验证 pod 是否已创建：

```
$ oc get pod
```

输出示例

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

6. 运行以下命令登录到 pod：

```
$ oc rsh tunepod
```

7. 验证配置的 sysctl 标记的值。例如，通过运行以下命令查找 net.ipv4.conf.net1.accept_redirects 的值：

```
sh-4.4# sysctl net.ipv4.conf.net1.accept_redirects
```

-

预期输出

```
net.ipv4.conf.net1.accept_redirects = 1
```

18.2. 使用调优 CNI 启用 ALL-MULTICAST 模式

您可以使用 tuning Container Network Interface (CNI) meta 插件启用 all-multicast 模式。

以下流程描述了如何配置调优 CNI 来启用 all-multicast 模式。

流程

1. 使用以下内容创建网络附加定义，如 tuning-example.yaml ：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: <name> 1
 namespace: default 2
spec:
 config: '{
  "cniVersion": "0.4.0", 3
  "name": "<name>", 4
  "plugins": [{
    "type": "<main_CNI_plugin>" 5
   },
   {
    "type": "tuning", 6
    "allmulti": true 7
   }
  }
 ]
}
```

1. 指定要创建的额外网络附加的名称。名称在指定的命名空间中必须是唯一的。
2. 指定与对象关联的命名空间。
3. 指定 CNI 规格版本。
4. 指定配置的名称。将配置名称与网络附加定义的 name 值匹配。
5. 指定要配置的主 CNI 插件的名称。
6. 指定 CNI meta 插件的名称。
7. 更改接口的 all-multicast 模式。如果启用，接口将接收网络上的所有多播数据包。

下面显示了一个 YAML 文件示例：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
```

```
metadata:
  name: setallmulti
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "setallmulti",
    "plugins": [
      {
        "type": "bridge"
      },
      {
        "type": "tuning",
        "allmulti": true
      }
    ]
  }'
```

2. 运行以下命令应用 YAML 文件中指定的设置：

```
$ oc apply -f tuning-allmulti.yaml
```

输出示例

```
networkattachmentdefinition.k8s.cni.cncf.io/setallmulti created
```

3. 使用类似以下 examplepod.yaml文件中指定的网络附加定义创建 pod ：

```
apiVersion: v1
kind: Pod
metadata:
  name: allmultipod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: setallmulti (1)
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 (2)
      runAsGroup: 3000 (3)
      allowPrivilegeEscalation: false (4)
      capabilities: (5)
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true (6)
    seccompProfile: (7)
      type: RuntimeDefault
```

(1) 指定配置的 NetworkAttachmentDefinition 的名称。

2 指定运行容器的用户 ID。

3 指定容器使用哪个主要组 ID。

4 指定 pod 是否可以请求特权升级。如果未指定，则默认为 true。这个布尔值直接控制在容器进程中是否设置了 no_new_privs 标志。

5 指定容器功能。drop: ["ALL"] 语句表示所有 Linux 功能都会从 pod 中丢弃，提供更严格的安全配置集。

6 指定容器将使用任何 UID 为 0 的用户运行。

7 指定容器的 seccomp 配置集。在这种情况下，type 被设置为 RuntimeDefault。seccomp 是一个 Linux 内核功能，它限制了进程可用的系统调用，通过最小化攻击面来提高安全性。

4. 运行以下命令应用 YAML 文件中指定的设置：

```
$ oc apply -f examplepod.yaml
```

5. 运行以下命令验证 pod 是否已创建：

```
$ oc get pod
```

输出示例

```
NAME        READY   STATUS    RESTARTS   AGE
allmultipod   1/1     Running   0          23s
```

6. 运行以下命令登录到 pod：

```
$ oc rsh allmultipod
```

7. 运行以下命令，列出与 pod 关联的所有接口：

```
sh-4.4# ip link
```

输出示例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue
state UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0 1
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0 2
```

1 eth0@if22 是主接口

2 net1@if24 是配置了 network-attachment-definition 的二级接口，它支持 all-multicast 模式 (ALLMULTI 标志)

18.3. 其他资源

- 在容器中使用 sysctl
- SR-IOV 网络节点配置对象
- 为 SR-IOV 网络配置接口级网络 sysctl 设置和 all-multicast 模式

第 19 章 在裸机集群中使用流控制传输协议 (SCTP)

作为集群管理员，您可以使用集群中的流控制传输协议 (SCTP)。

19.1. 支持 OPENSHIFT CONTAINER PLATFORM 上的流控制传输协议 (SCTP)

作为集群管理员，您可以在集群中的主机上启用 SCTP。在 Red Hat Enterprise Linux CoreOS (RHCOS) 上，SCTP 模块被默认禁用。

SCTP 是基于信息的可靠协议，可在 IP 网络之上运行。

启用后，您可以使用 SCTP 作为带有 pod、服务和网络策略的协议。**Service** 对象必须通过将 **type** 参数设置为 **ClusterIP** 或 **NodePort** 值来定义。

19.1.1. 使用 SCTP 协议的示例配置

您可以通过将 pod 或服务对象中的 **protocol** 参数设置为 **SCTP** 来将 pod 或服务配置为使用 SCTP。

在以下示例中，pod 被配置为使用 SCTP:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
      ports:
        - containerPort: 30100
          name: sctpserver
          protocol: SCTP
```

在以下示例中，服务被配置为使用 SCTP:

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
  type: ClusterIP
```

在以下示例中，**NetworkPolicy** 对象配置为对来自具有特定标签的任何 pod 的端口 80 应用 SCTP 网络流量：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
    - protocol: SCTP
      port: 80
```

19.2. 启用流控制传输协议 (SCTP)

作为集群管理员，您可以在集群中的 worker 节点上加载并启用列入黑名单的 SCTP 内核模块。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建名为 load-sctp-module.yaml 的文件，其包含以下 YAML 定义：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: load-sctp-module
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modprobe.d/sctp-blacklist.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,
        - path: /etc/modules-load.d/sctp-load.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,sctp
```

2. 运行以下命令来创建 MachineConfig 对象：

```
$ oc create -f load-sctp-module.yaml
```

3. 可选：要在 MachineConfig Operator 应用配置更改时监测节点的状态，请使用以下命令。当节点状态变为 Ready时，则代表配置更新已被应用。

```
$ oc get nodes
```

19.3. 验证流控制传输协议 (SCTP) 已启用

您可以通过创建一个 pod 以及侦听 SCTP 流量的应用程序，将其与服务关联，然后连接到公开的服务，来验证 SCTP 是否在集群中工作。

先决条件

- 从集群访问互联网来安装 nc 软件包。
- 安装 OpenShift CLI (oc)。
- 使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建 pod 启动 SCTP 侦听程序：
   a. 创建名为 sctp-server.yaml 的文件，该文件使用以下 YAML 定义 pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

   b. 运行以下命令来创建 pod：

```
$ oc create -f sctp-server.yaml
```

2. 为 SCTP 侦听程序 pod 创建服务。
   a. 创建名为 sctp-service.yaml 的文件，该文件使用以下 YAML 定义服务：

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
```

```
    labels:
      app: sctpserver
  spec:
    type: NodePort
    selector:
      app: sctpserver
    ports:
      - name: sctpserver
        protocol: SCTP
        port: 30102
        targetPort: 30102
```

b. 要创建服务，请输入以下命令：

```
$ oc create -f sctp-service.yaml
```

3. 为 SCTP 客户端创建 pod。

a. 使用以下 YAML 创建名为 sctp-client.yaml 的文件：

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
```

b. 运行以下命令来创建 Pod 对象：

```
$ oc apply -f sctp-client.yaml
```

4. 在服务器中运行 SCTP 侦听程序。

a. 要连接到服务器 pod，请输入以下命令：

```
$ oc rsh sctpserver
```

b. 要启动 SCTP 侦听程序，请输入以下命令：

```
$ nc -l 30102 --sctp
```

5. 连接到服务器上的 SCTP 侦听程序。

a. 在终端程序里打开一个新的终端窗口或标签页。

b. 获取 sctpservice 服务的 IP 地址。使用以下命令：

```
$ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
```

c. 要连接到客户端 pod，请输入以下命令：

```
$ oc rsh sctpclient
```

d. 要启动 SCTP 客户端，请输入以下命令。将 <cluster_IP> 替换为 sctpservice 服务的集群 IP 地址。

```
# nc <cluster_IP> 30102 --sctp
```

第 20 章 使用 PTP 硬件

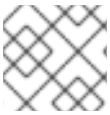
20.1. 关于 OPENSHIFT CONTAINER PLATFORM 集群节点中的 PTP

精度时间协议(PTP)用于同步网络中的时钟。与硬件支持一起使用时，PTP 能够达到微秒级的准确性，比网络时间协议 (NTP) 更加准确。

您可以配置 linuxptp 服务，并在 OpenShift Container Platform 集群节点中使用具有 PTP 功能的硬件。

通过部署 PTP Operator，使用 OpenShift Container Platform Web 控制台或 OpenShift CLI (oc)安装 PTP。PTP Operator 会创建和管理 linuxptp 服务，并提供以下功能：

- 在集群中发现具有 PTP 功能的设备。
- 管理 linuxptp 服务的配置。
- PTP 时钟事件通知会使用 PTP Operator cloud-event-proxy sidecar 会对应用程序的性能和可靠性造成负面影响。



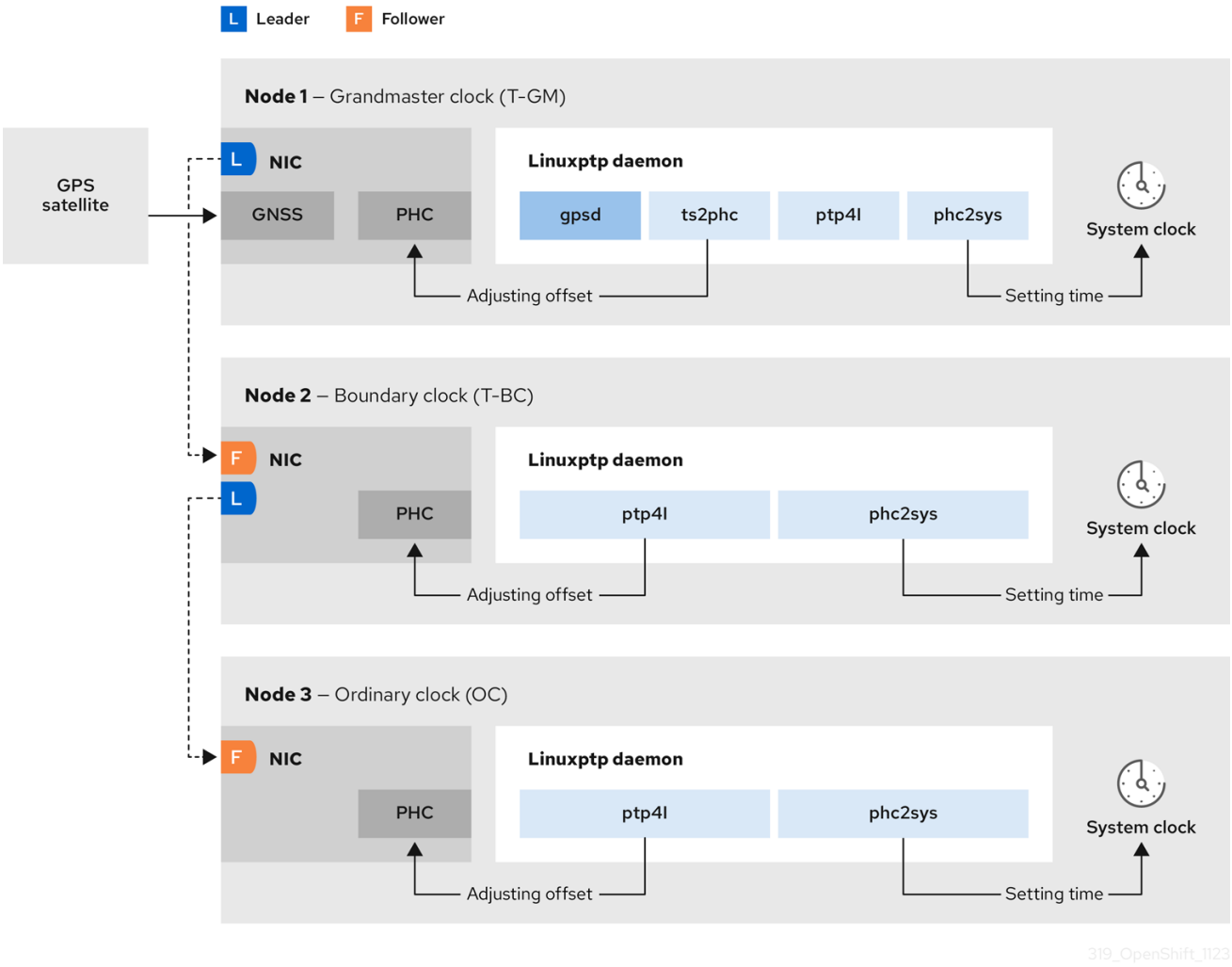
注意

PTP Operator 只适用于仅在裸机基础架构上置备的集群上具有 PTP 功能的设备。

20.1.1. PTP 域的元素

PTP 用于将网络中连接的多个节点与每个节点的时钟同步。PTP 同步时钟以领导层次结构进行组织。层次结构由最佳 master 时钟 (BMC) 算法自动创建和更新，该算法在每个时钟上运行。后续时钟与领导时钟同步，后续时钟本身可以是其他下游时钟的来源。

图 20.1. 网络中的 PTP 节点



下面描述了三种 PTP 时钟类型。

Grandmaster 时钟

grandmaster 时钟向网络上的其他时钟提供标准时间信息并确保准确和稳定的同步。它写入时间戳并响应来自其他时钟的时间间隔。grandmaster 时钟与全局导航 Satellite 系统 (GNSS) 时间源同步。Grandmaster 时钟是网络中权威时间来源，负责为所有其他设备提供时间同步。

Boundary 时钟

Boundary（边界）时钟在两个或更多个通信路径中具有端口，并且可以是指向其他目标时钟的源和目标。边界时钟作为上游目标时钟工作。目标时钟接收计时消息，针对延迟进行调整，然后创建一个新的源时间信号来传递网络。边界时钟生成一个新的计时数据包，它仍然与源时钟正确同步，并可减少直接报告到源时钟的连接设备数量。

Ordinary 时钟

Ordinary（普通）时钟具有一个端口连接，可根据其在网络中的位置扮演源或目标时钟的角色。普通时钟可以读取和写入时间戳。

PTP 优于 NTP 的优点

PTP 与 NTP 相比有一个主要优势，即各种网络接口控制器 (NIC) 和网络交换机中存在的硬件支持。特殊硬件允许 PTP 考虑消息传输的延迟，并提高时间同步的准确性。为了获得最佳准确性，建议启用 PTP 时钟间的所有网络组件。

基于硬件的 PTP 提供最佳准确性，因为 NIC 可以在准确发送和接收时对 PTP 数据包进行时间戳。这与基于软件的 PTP 进行比较，这需要操作系统对 PTP 数据包进行额外的处理。

重要

在启用 PTP 前，请确保为所需节点禁用 NTP。您可以使用 MachineConfig 自定义资源禁用 chrony 时间服务 (chronyd)。如需更多信息，请参阅禁用 chrony 时间服务。

20.1.2. 使用带有 PTP 的双 NIC Intel E810 硬件

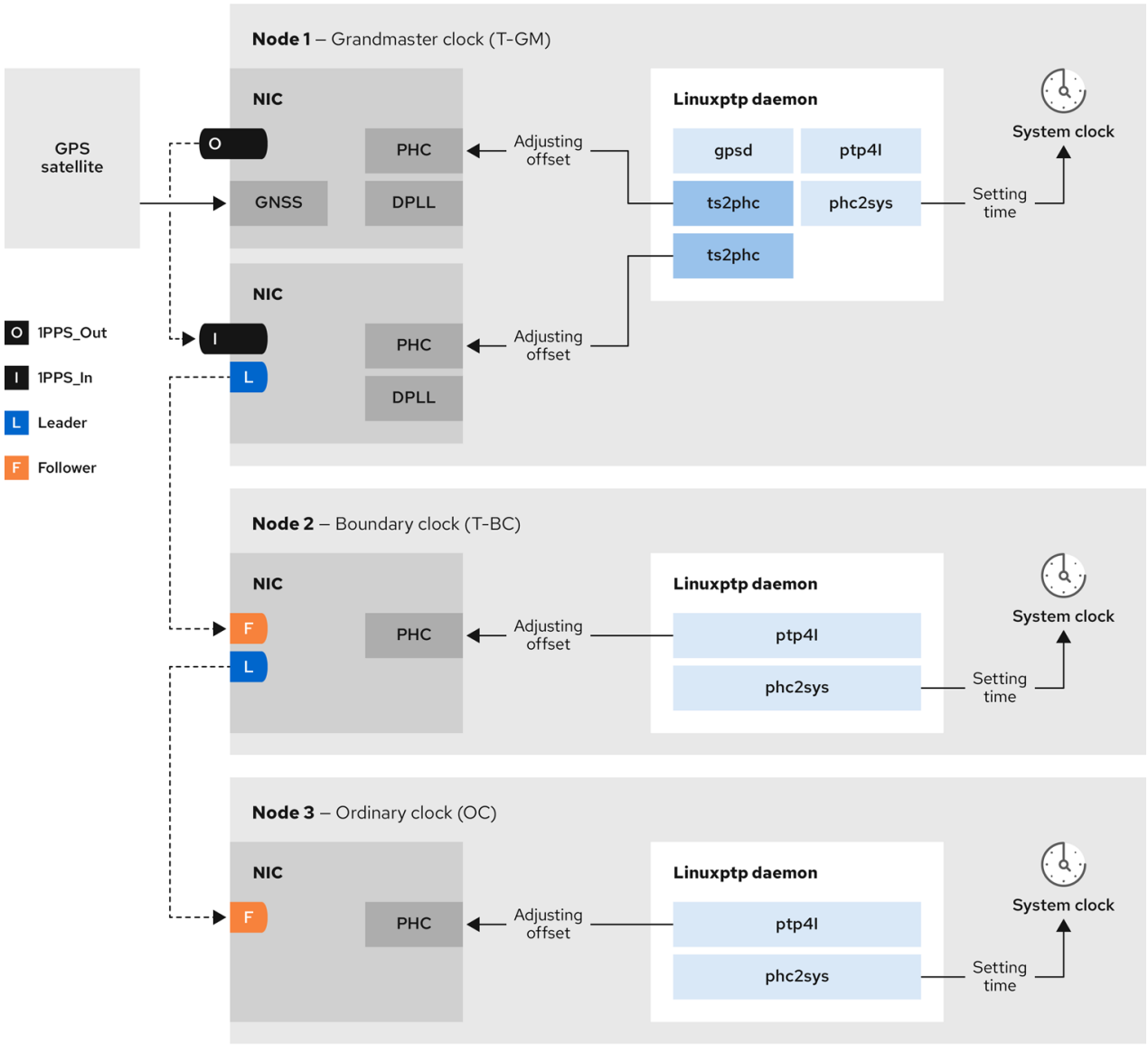
OpenShift Container Platform 支持单和双 NIC Intel E810 硬件，以便在 grandmaster 时钟 (T-GM) 和边界时钟 (T-BC) 中进行精度 PTP 时间。

双 NIC grandmaster 时钟

您可以使用具有双 NIC 硬件作为 PTP grandmaster 时钟的集群主机。一个 NIC 从全局导航 Satellite 系统(GNSS)接收计时信息。第二个 NIC 在第一次使用 E810 NIC 上的 SMA1 Tx/Rx 连接接收时间信息。集群主机上的系统时钟从连接到 GNSS satellite 的 NIC 同步。

双 NIC grandmaster 时钟是分布式 RAN (D-RAN)配置的功能，其中远程 Radio 单元(RRU)和 Baseband 单元(BBU)位于相同的无线单元站点。d-RAN 在多个站点间分发无线功能，带有将它们链接到核心网络的连接。

图 20.2. 双 NIC grandmaster 时钟



注意

在双 NIC T-GM 配置中，单个 ts2phc 进程报告为系统中的两个 ts2phc 实例。

双 NIC 边界时钟

对于提供中等范围的 5G 电信网络，每个虚拟分布式单元(vDU)需要连接到 6 个无线电单元(RU)。要使这些连接，每个 vDU 主机都需要 2 个 NIC 被配置为边界时钟。
双 NIC 硬件允许您将每个 NIC 连接到相同的上游领导时钟，并将每个 NIC 的 ptp4l 实例连接给下游时钟。

带有双 NIC 边界时钟的高可用性系统时钟

您可以将 Intel E810-XXVDA4 Salem 频道双 NIC 硬件配置为双 PTP 边界时钟，为高可用性系统时钟提供计时。当您在不同 NIC 上有多个时间源时，这非常有用。高可用性可确保如果两个计时源丢失或断开连接，则节点不会丢失计时同步。
每个 NIC 都连接到同一上游领导时钟。高可用性边界时钟使用多个 PTP 域与目标系统时钟同步。当 T-BC 高度可用时，主机系统时钟可以维护正确的偏移，即使一个或多个 ptp4l 实例同步 NIC PHC 时钟失败。如果发生任何单一 SFP 端口或电缆失败，则边界时钟会与领导时钟保持同步。

边界时钟领导源选择使用 A-BMCA 算法完成。如需更多信息，请参阅 ITU-T 建议 G.8275.1。

20.1.3. OpenShift Container Platform 节点中的 linuxptp 和 gpsd 概述

OpenShift Container Platform 使用带有 linuxptp 和 gpsd 软件包的 PTP Operator 进行高精度网络同步。linuxptp 软件包为网络中的 PTP 时间提供工具和守护进程。带有 Global Navigation Satellite System (GNSS)功能 NIC 的集群主机使用 gpsd 来与 GNSS 时钟源进行接口。

linuxptp 软件包包括用于系统时钟同步的 ts2phc、pmc、ptp4l 和 phc2sys 程序。

ts2phc

ts2phc 将 PTP 设备中的 PTP 硬件时钟(PHC)与高度精确度同步。ts2phc 用于 grandmaster 时钟配置。它收到精度计时信号，这是一个高度精确时钟源，如 Global Navigation Satellite System (GNSS)。GNSS 提供准确可靠的同步时间源，用于大型分布式网络。GNSS 时钟通常提供时间信息，其精度为几个纳秒。
ts2phc 系统守护进程通过读取 grandmaster 时钟中的时间信息，将时间信息从 grandmaster 时钟发送到网络中的其他 PTP 设备，并将其转换为 PHC 格式。PHC 时间供网络中的其他设备用来将其时钟与 grandmaster 时钟同步。

pmc

pmc 根据 IEEE 标准 1588.1588 实现 PTP 管理客户端 (pmc)。pmc 为 ptp4l 系统守护进程提供基本的管理访问权限。pmc 从标准输入读取，并通过所选传输发送输出，打印它收到的任何回复。

ptp4l

ptp4l 实现 PTP 边界时钟和普通时钟，并作为系统守护进程运行。ptp4l 执行以下操作：

- 将 PHC 同步到源时钟与硬件时间戳
- 将系统时钟与源时钟与软件时间戳同步

phc2sys

phc2sys 将系统时钟与网络接口控制器 (NIC) 上的 PHC 同步。phc2sys 系统守护进程持续监控 PHC 以获取计时信息。当检测到计时错误时，LareC 会更正系统时钟。

gpsd 软件包包括 ubxtool、gspipe、gpsd、GNSS 时钟与主机时钟同步的程序。

ubxtool

ubxtool CLI 可让您与 u-blox GPS 系统通信。ubxtool CLI 使用 u-blox 二进制协议与 GPS 通信。

gpspipe

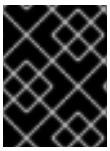
gpspipe 连接到 gpsd 输出并将其传送到 stdout。

gpsd

gpsd 是一个服务守护进程，它监控一个或多个连接到主机的 GPS 或 AIS 接收器。

20.1.4. PTP grandmaster 时钟的 GNSS 时间概述

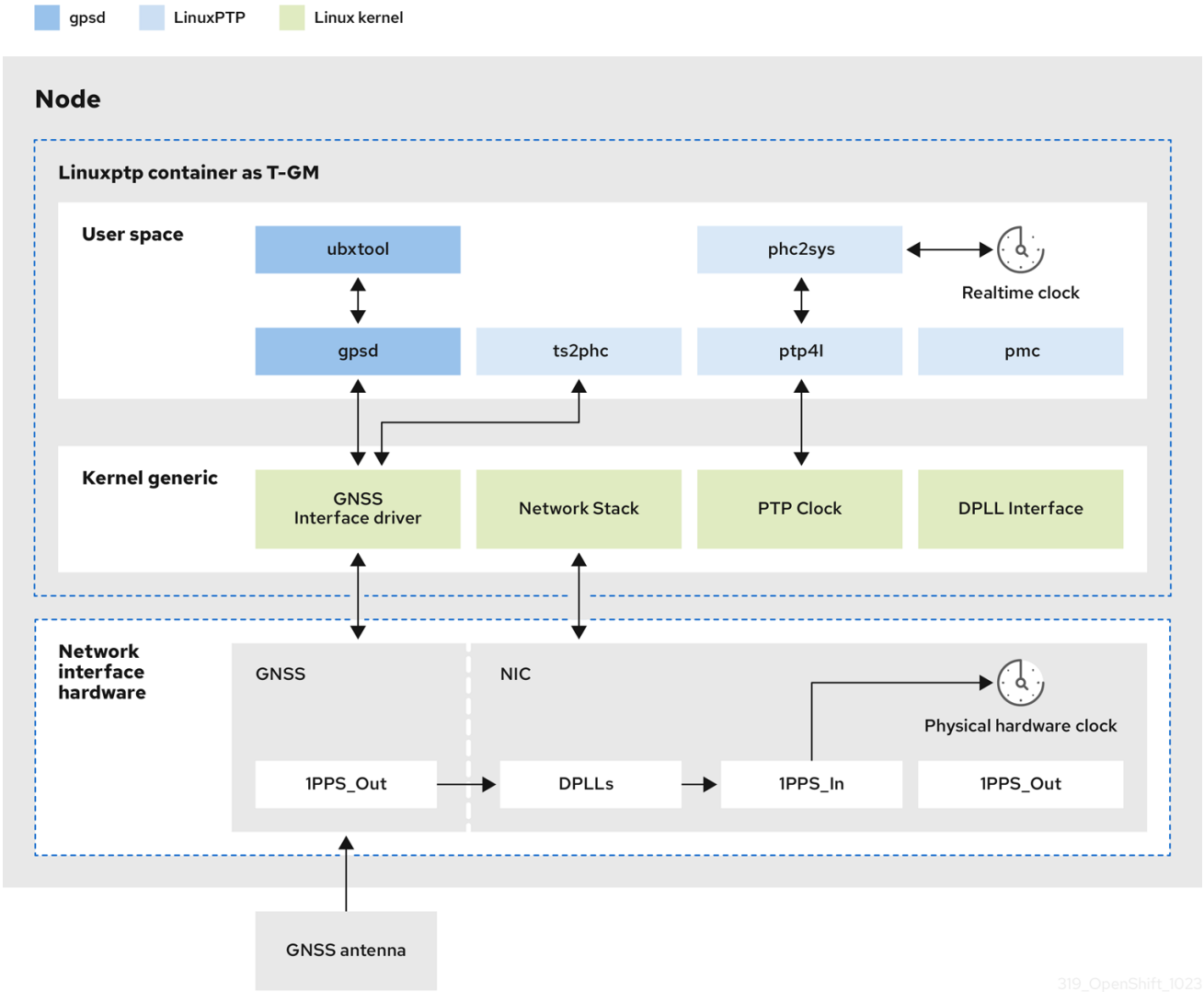
OpenShift Container Platform 支持从集群中的 Global Navigation Satellite 系统(GNSS)源和 grandmaster 时钟(T-GM)接收精度 PTP 时间。



重要

OpenShift Container Platform 仅支持 Intel E810 Westport Channel NIC 的 GNSS 源中的 PTP 时间。

图 20.3. 使用 GNSS 和 T-GM 同步概述



全局导航 Satellite 系统(GNSS)

GNSS 是一个基于 satellite 的系统，用来为全球范围内接收器提供定位、导航和计时信息。在 PTP 中，GNSS 接收器通常用作高度准确且稳定的参考时钟源。这些接收器从多个 GNSS satellites 接收信号，允许它们计算精确的时间信息。从 GNSS 获取的时间信息被 PTP grandmaster 时钟参考。通过将 GNSS 用作参考，PTP 网络中的 grandmaster 时钟可以为其他设备提供高度准确的时间戳，从而在整个网络中启用精确同步。

Digital Phase-Locked Loop (DPLL)

DPLL 在网络中的不同 PTP 节点之间提供时钟同步。DPLL 将本地系统时钟信号的阶段与传入同步信号的阶段进行比较，例如，来自 PTP grandmaster 时钟的 PTP 信息。DPLL 持续调整本地时钟频率和阶段，以最大程度降低本地时钟和参考时钟之间的阶段差异。

20.2. 配置 PTP 设备

PTP Operator 将 **NodePtpDevice.ptp.openshift.io** 自定义资源定义（CRD）添加到 OpenShift Container Platform。

安装后，PTP Operator 会在每个节点中搜索具有 PTP 功能的网络设备。它为提供兼容 PTP 的网络设备的每个节点创建并更新 **NodePtpDevice** 自定义资源(CR)对象。

20.2.1. 使用 CLI 安装 PTP Operator

作为集群管理员，您可以使用 CLI 安装 Operator。

先决条件

- 在裸机中安装有支持 PTP 硬件的节点的集群。
- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 为 PTP Operator 创建命名空间。
   a. 将以下 YAML 保存到 **ptp-namespace.yaml** 文件中：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
```

   b. 创建 **Namespace** CR：

```
$ oc create -f ptp-namespace.yaml
```

2. 为 PTP Operator 创建 Operator 组。

a. 在 ptp-operatorgroup.yaml 文件中保存以下 YAML：

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
```

b. 创建 OperatorGroup CR：

```
$ oc create -f ptp-operatorgroup.yaml
```

3. 订阅 PTP Operator。

a. 将以下 YAML 保存到 ptp-sub.yaml 文件中：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. 创建 Subscription CR：

```
$ oc create -f ptp-sub.yaml
```

4. 要验证是否已安装 Operator，请输入以下命令：

```
$ oc get csv -n openshift-ptp -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

输出示例

```
Name                    Phase
4.16.0-202301261535     Succeeded
```

20.2.2. 使用 Web 控制台安装 PTP Operator

作为集群管理员，您可以使用 Web 控制台安装 PTP Operator。

注意

如上一节所述，您必须创建命名空间和 operator 组。

流程

1. 使用 OpenShift Container Platform Web 控制台安装 PTP Operator：
   a. 在 OpenShift Container Platform Web 控制台中，点击 Operators → OperatorHub。
   b. 从可用的 Operator 列表中选择 PTP Operator，然后点 Install。
   c. 在 Install Operator 页面中，在 A specific namespace on the cluster下选择 openshift-ptp。然后点击 Install。
2. 可选：验证是否成功安装了 PTP Operator：
   a. 切换到 Operators → Installed Operators 页面。
   b. 确保 openshift-ptp 项目中列出的 PTP Operator 的 Status 为 InstallSucceeded。

注意

在安装过程中，Operator 可能会显示 Failed 状态。如果安装过程结束后有 InstallSucceeded 信息，您可以忽略这个 Failed 信息。

如果 Operator 没有被成功安装，请按照以下步骤进行故障排除：

- 进入 Operators → Installed Operators 页面，检查 Operator Subscriptions 和 Install Plans 选项卡中的 Status 项中是否有任何错误。
- 进入 Workloads → Pods 页面，检查 **openshift-ptp** 项目中 pod 的日志。

20.2.3. 在集群中发现具有 PTP 功能网络设备

- 要返回集群中具有 PTP 功能网络设备的完整列表，请运行以下命令：

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0 (1)
    namespace: openshift-ptp
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices: (2)
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
```

```
    - name: enp5s0f0
    - name: enp5s0f1
...
```

1. name 参数的值与父节点的名称相同。
2. devices 集合包含 PTP Operator 发现节点的 PTP 功能设备列表。

20.2.4. 在 PTP Operator 中使用特定于硬件的 NIC 功能

带有内置 PTP 功能的 NIC 硬件有时需要特定于设备的配置。您可以通过在 PtpConfig 自定义资源(CR)中配置插件，将特定于硬件的 NIC 功能用于 PTP Operator 支持的硬件。linuxptp-daemon 服务使用 plugin 小节中的指定参数根据特定的硬件配置启动 linuxptp 进程(ptp4l 和 phc2sys)。

重要

在 OpenShift Container Platform 4.16 中，通过 PtpConfig 插件支持 Intel E810 NIC。

20.2.5. 将 linuxptp 服务配置为 grandmaster 时钟

您可以通过创建一个配置主机 NIC 的 PtpConfig 自定义资源(CR)将 linuxptp 服务(ptp4l、phc2sys、ts2phc)配置为 grandmaster 时钟(T-GM)。

ts2phc 工具允许您将系统时钟与 PTP grandmaster 时钟同步，以便节点可以将精度时钟信号流传输到下游 PTP 普通时钟和边界时钟。

注意

使用 PtpConfig CR 示例，将 linuxptp 服务配置为 Intel Westport Channel E810-XXVDA4T 网络接口的 T-GM。

要配置 PTP 快速事件，请为 ptp4lOpts、ptp4lConf 和 ptpClockThreshold 设置适当的值。ptpClockThreshold 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 对于生产环境中的 T-GM 时钟，请在裸机集群主机上安装 Intel E810 Westport Channel NIC。
- 安装 OpenShift CLI (oc) 。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建 PtpConfig CR。例如：
   a. 根据您的要求，为您的部署使用以下 T-GM 配置之一。将 YAML 保存到 grandmaster-clock-ptp-config.yaml 文件中：

      例 20.1. PTP grandmaster 时钟配置示例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalMaxHoldoverOffSet: 1500
          LocalHoldoverTimeout: 14400
          MaxInSpecOffset: 100
        pins: $e810_pins
        #  "$iface_master":
        #    "U.FL2": "0 2"
        #    "U.FL1": "0 1"
        #    "SMA2": "0 2"
        #    "SMA1": "0 1"
        ublxCmds:
          - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
              - "-P"
              - "29.20"
              - "-z"
              - "CFG-HW-ANT_CFG_VOLTCTRL,1"
            reportOutput: false
          - args: #ubxtool -P 29.20 -e GPS
              - "-P"
              - "29.20"
              - "-e"
              - "GPS"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d Galileo
              - "-P"
              - "29.20"
              - "-d"
              - "Galileo"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d GLONASS
              - "-P"
              - "29.20"
              - "-d"
              - "GLONASS"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d BeiDou
              - "-P"
              - "29.20"
              - "-d"
```

```
        - "BeiDou"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
        - "-P"
        - "29.20"
        - "-t"
        - "-w"
        - "5"
        - "-v"
        - "1"
        - "-e"
        - "SURVEYIN,600,50000"
      reportOutput: true
    - args: #ubxtool -P 29.20 -p MON-HW
        - "-P"
        - "29.20"
        - "-p"
        - "MON-HW"
      reportOutput: true
ts2phcOpts: " "
ts2phcConf: |
  [nmea]
  ts2phc.master 1
  [global]
  use_syslog  0
  verbose 1
  logging_level 7
  ts2phc.pulsewidth 100000000
  ts2phc.nmea_serialport $gnss_serialport
  leapfile  /usr/share/zoneinfo/leap-seconds.list
  [$iface_master]
  ts2phc.extts_polarity rising
  ts2phc.extts_correction 0
ptp4lConf: |
  [$iface_master]
  masterOnly 1
  [$iface_master_1]
  masterOnly 1
  [$iface_master_2]
  masterOnly 1
  [$iface_master_3]
  masterOnly 1
  [global]
  #
  # Default Data Set
  #
  twoStepFlag 1
  priority1 128
  priority2 128
  domainNumber 24
```

```
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
```

```
    step_threshold 2.0
    first_step_threshold 0.00002
    clock_servo pi
    sanity_freq_limit  200000000
    ntpshm_segment 0
    #
    # Transport options
    #
    transportSpecific 0x0
    ptp_dst_mac 01:1B:19:00:00:00
    p2p_dst_mac 01:80:C2:00:00:0E
    udp_ttl 1
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type BC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0x20
  recommend:
  - profile: "grandmaster"
    priority: 4
    match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"
```

注意

PTP grandmaster 时钟配置示例仅用于测试目的，不适用于生产环境。

例 20.2. E810 NIC 的 PTP grandmaster 时钟配置

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
spec:
```

```
profile:
- name: "grandmaster"
  ptp4lOpts: "-2 --summary_interval -4"
  phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
  ptpSchedulingPolicy: SCHED_FIFO
  ptpSchedulingPriority: 10
  ptpSettings:
    logReduce: "true"
  plugins:
    e810:
      enableDefaultConfig: false
      settings:
        LocalMaxHoldoverOffSet: 1500
        LocalHoldoverTimeout: 14400
        MaxInSpecOffset: 100
      pins: $e810_pins
      #  "$iface_master":
      #    "U.FL2": "0 2"
      #    "U.FL1": "0 1"
      #    "SMA2": "0 2"
      #    "SMA1": "0 1"
      ublxCmds:
        - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
            - "-P"
            - "29.20"
            - "-z"
            - "CFG-HW-ANT_CFG_VOLTCTRL,1"
          reportOutput: false
        - args: #ubxtool -P 29.20 -e GPS
            - "-P"
            - "29.20"
            - "-e"
            - "GPS"
          reportOutput: false
        - args: #ubxtool -P 29.20 -d Galileo
            - "-P"
            - "29.20"
            - "-d"
            - "Galileo"
          reportOutput: false
        - args: #ubxtool -P 29.20 -d GLONASS
            - "-P"
            - "29.20"
            - "-d"
            - "GLONASS"
          reportOutput: false
        - args: #ubxtool -P 29.20 -d BeiDou
            - "-P"
            - "29.20"
            - "-d"
            - "BeiDou"
          reportOutput: false
        - args: #ubxtool -P 29.20 -d SBAS
            - "-P"
            - "29.20"
            - "-d"
```

```
          - "SBAS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
          - "-P"
          - "29.20"
          - "-t"
          - "-w"
          - "5"
          - "-v"
          - "1"
          - "-e"
          - "SURVEYIN,600,50000"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p MON-HW
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      ts2phc.nmea_serialport $gnss_serialport
      leapfile  /usr/share/zoneinfo/leap-seconds.list
      [$iface_master]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
    ptp4lConf: |
      [$iface_master]
      masterOnly 1
      [$iface_master_1]
      masterOnly 1
      [$iface_master_2]
      masterOnly 1
      [$iface_master_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 6
      clockAccuracy 0x27
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
```

```
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit  200000000
ntpshm_segment 0
#
```

```
    # Transport options
    #
    transportSpecific 0x0
    ptp_dst_mac 01:1B:19:00:00:00
    p2p_dst_mac 01:80:C2:00:00:0E
    udp_ttl 1
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type BC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0x20
recommend:
- profile: "grandmaster"
  priority: 4
  match:
  - nodeLabel: "node-role.kubernetes.io/$mcp"
```

注意

对于 E810 Westport Channel NIC，将 **ts2phc.nmea_serialport** 的值设置为 **/dev/gnss0**。

b. 运行以下命令来创建 CR：

```
$ oc create -f grandmaster-clock-ptp-config.yaml
```

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   输出示例

```
NAME                           READY   STATUS    RESTARTS   AGE    IP           NODE
linuxptp-daemon-74m2g          3/3     Running   3          4d15h  10.16.230.7
compute-1.example.com
ptp-operator-5f4f48d7c-x7zkf   1/1     Running   1          4d15h  10.128.1.145
compute-1.example.com
```

b. 检查配置集是否正确。检查与 PtpConfig 配置集中指定的节点对应的 linuxptp 守护进程的日志。运行以下命令:

```
$ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
```

输出示例

```
ts2phc[94980.334]: [ts2phc.0.config] nmea delay: 98690975 ns
ts2phc[94980.334]: [ts2phc.0.config] ens3f0 extts index 0 at 1676577329.999999999
corr 0 src 1676577330.901342528 diff -1
ts2phc[94980.334]: [ts2phc.0.config] ens3f0 master offset         -1 s2 freq      -1
ts2phc[94980.441]: [ts2phc.0.config] nmea sentence:
GNRMC,195453.00,A,4233.24427,N,07126.64420,W,0.008,,160223,,,A,V
phc2sys[94980.450]: [ptp4l.0.config] CLOCK_REALTIME phc offset       943 s2 freq
-89604 delay    504
phc2sys[94980.512]: [ptp4l.0.config] CLOCK_REALTIME phc offset      1000 s2 freq
-89264 delay    474
```

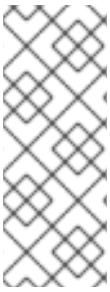
20.2.6. 将 linuxptp 服务配置为双 E810 Westport Channel NIC 的 grandmaster 时钟

您可以通过创建一个 PtpConfig 自定义资源(CR)来为双 E810 Westport Channel NIC 将linuxptp 服务(ptp4l、phc2sys、ts2phc)配置为 grandmaster 时钟(T-GM)。

对于分布式 RAN (D-RAN)用例，您可以为双 NIC 配置 PTP，如下所示：

- NIC 一个与全局导航 Satellite 系统(GNSS)时间源同步。
- NIC 2 将同步到 NIC 提供的 1PPS 时间输出。此配置由 PtpConfig CR 中的 PTP 硬件插件提供。

双 NIC PTP T-GM 配置使用单个 ptp4l 实例，一个 ts2phc 进程报告两个 ts2phc 实例，每个 NIC 都有一个 ts2phc 实例。主机系统时钟与连接到 GNSS 时间源的 NIC 同步。



注意

使用 PtpConfig CR 示例，将linuxptp 服务配置为双 Intel Westport Channel E810-XXVDA4T 网络接口的 T-GM。

要配置 PTP 快速事件，请为 ptp4lOpts、ptp4lConf 和 ptpClockThreshold 设置适当的值。ptpClockThreshold 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 对于生产环境中的 T-GM 时钟，请在裸机集群主机上安装两个 Intel E810 Westport Channel NIC。
- 安装 OpenShift CLI (oc) 。

- 以具有 cluster-admin 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建 **PtpConfig** CR。例如：
   a. 将以下 YAML 保存到 **grandmaster-clock-ptp-config-dual-nics.yaml** 文件中：

例 20.3. 用于双 E810 NIC 的 PTP grandmaster 时钟配置

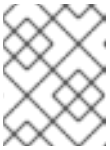
```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_nic1 -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalMaxHoldoverOffSet: 1500
          LocalHoldoverTimeout: 14400
          MaxInSpecOffset: 100
        pins:
          "$iface_nic1":
            "U.FL2": "0 2"
            "U.FL1": "0 1"
            "SMA2": "0 2"
            "SMA1": "2 1"
          "$iface_nic2":
            "U.FL2": "0 2"
            "U.FL1": "0 1"
            "SMA2": "0 2"
            "SMA1": "1 1"
        ublxCmds:
          - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
              - "-P"
              - "29.20"
              - "-z"
              - "CFG-HW-ANT_CFG_VOLTCTRL,1"
            reportOutput: false
          - args: #ubxtool -P 29.20 -e GPS
              - "-P"
              - "29.20"
              - "-e"
```

```
        - "GPS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d Galileo
        - "-P"
        - "29.20"
        - "-d"
        - "Galileo"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d GLONASS
        - "-P"
        - "29.20"
        - "-d"
        - "GLONASS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d BeiDou
        - "-P"
        - "29.20"
        - "-d"
        - "BeiDou"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
        - "-P"
        - "29.20"
        - "-t"
        - "-w"
        - "5"
        - "-v"
        - "1"
        - "-e"
        - "SURVEYIN,600,50000"
      reportOutput: true
    - args: #ubxtool -P 29.20 -p MON-HW
        - "-P"
        - "29.20"
        - "-p"
        - "MON-HW"
      reportOutput: true
  ts2phcOpts: " "
  ts2phcConf: |
    [nmea]
    ts2phc.master 1
    [global]
    use_syslog  0
    verbose 1
    logging_level 7
    ts2phc.pulsewidth 100000000
    #cat /dev/GNSS to find available serial port
    #example value of gnss_serialport is /dev/ttyGNSS_1700_0
    ts2phc.nmea_serialport $gnss_serialport
    leapfile  /usr/share/zoneinfo/leap-seconds.list
```

```
      [$iface_nic1]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
      [$iface_nic2]
      ts2phc.master 0
      ts2phc.extts_polarity rising
      #this is a measured value in nanoseconds to compensate for SMA cable
delay
      ts2phc.extts_correction -10
    ptp4lConf: |
      [$iface_nic1]
      masterOnly 1
      [$iface_nic1_1]
      masterOnly 1
      [$iface_nic1_2]
      masterOnly 1
      [$iface_nic1_3]
      masterOnly 1
      [$iface_nic2]
      masterOnly 1
      [$iface_nic2_1]
      masterOnly 1
      [$iface_nic2_2]
      masterOnly 1
      [$iface_nic2_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 6
      clockAccuracy 0x27
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
      dscp_event 0
      dscp_general 0
      dataset_comparison G.8275.x
      G.8275.defaultDS.localPriority 128
      #
      # Port Data Set
      #
      logAnnounceInterval -3
      logSyncInterval -4
      logMinDelayReqInterval -4
      logMinPdelayReqInterval 0
      announceReceiptTimeout 3
      syncReceiptTimeout 0
      delayAsymmetry 0
      fault_reset_interval -4
      neighborPropDelayThresh 20000000
```

```
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit  200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
```

```
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 1
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0x20
recommend:
- profile: "grandmaster"
  priority: 4
  match:
  - nodeLabel: "node-role.kubernetes.io/$mcp"
```



注意

对于 E810 Westport Channel NIC，将 **ts2phc.nmea_serialport** 的值设置为 **/dev/gnss0**。

b. 运行以下命令来创建 CR：

```
$ oc create -f grandmaster-clock-ptp-config-dual-nics.yaml
```

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   输出示例

   ```
   NAME                     READY   STATUS    RESTARTS   AGE    IP           NODE
   linuxptp-daemon-74m2g    3/3     Running   3          4d15h  10.16.230.7
   compute-1.example.com
   ptp-operator-5f4f48d7c-x7zkf  1/1     Running   1          4d15h  10.128.1.145
   compute-1.example.com
   ```

   b. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令：

   ```
   $ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
   ```

   输出示例

```
ts2phc[509863.660]: [ts2phc.0.config] nmea delay: 347527248 ns
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 extts index 0 at
1705516553.000000000 corr 0 src 1705516553.652499081 diff 0
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 master offset          0 s2 freq      -0
I0117 18:35:16.000146 1633226 stats.go:57] state updated for ts2phc =s2
I0117 18:35:16.000163 1633226 event.go:417] dpll State s2, gnss State s2, tsphc
state s2, gm state s2,
ts2phc[1705516516]:[ts2phc.0.config] ens2f0 nmea_status 1 offset 0 s2
GM[1705516516]:[ts2phc.0.config] ens2f0 T-GM-STATUS s2
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 extts index 0 at
1705516553.000000010 corr -10 src 1705516553.652499081 diff 0
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 master offset          0 s2 freq      -0
I0117 18:35:16.016597 1633226 stats.go:57] state updated for ts2phc =s2
phc2sys[509863.719]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -6 s2 freq
+15441 delay    510
phc2sys[509863.782]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -7 s2 freq
+15438 delay    502
```

其他资源

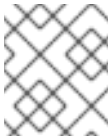
- 配置 PTP 快速事件通知发布程序

20.2.6.1. grandmaster clock PtpConfig 配置参考

以下参考信息描述了 PtpConfig 自定义资源(CR)的配置选项，将 linuxptp 服务 (ptp4l、phc2sys、ts2phc)配置为 grandmaster 时钟。

表 20.1. PTP Grandmaster 时钟的 PtpConfig 配置选项

PtpConfig CR 字段	描述
plugins	指定一组 **.exec.cmdline** 选项来为 grandmaster 时钟操作配置 NIC。grandmaster 时钟配置需要禁用某些 PTP pin。 插件机制允许 PTP Operator 进行自动硬件配置。对于 Intel Westport Channel NIC，当 **enableDefaultConfig** 为 true 时，PTP Operator 运行一个硬编码的脚本来为 NIC 执行所需的配置。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。
ptp4lConf	指定启动 **ptp4l** 作为 grandmaster 时钟所需的配置。例如，**ens2f1** 接口同步下游连接的设备。对于 grandmaster 时钟，将 **clockClass** 设置为 **6**，并将 **clockAccuracy** 设置为 **0x27**。将 **timeSource** 设置为 **0x20**，以便在从全局导航 Satellite 系统 (GNSS) 接收计时信号时。
tx_timestamp_timeout	指定丢弃数据前从发送方等待传输 (TX) 时间戳的最长时间。
boundary_clock_jbod	指定 JBOD 边界时钟时间延迟值。这个值用于更正网络时间设备之间传递的时间值。

PtpConfig CR 字段	描述
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。  **注意** 确保此处列出的网络接口配置为 grandmaster，并在 **ts2phcConf** 和 **ptp4lConf** 字段中根据需要引用。
ptpSchedulingPolicy	为 **ptp4l** 和 **phc2sys** 进程配置调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，设置 1-65 的整数值来为 **ptp4l** 和 **phc2sys** 进程配置 FIFO 优先级。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。
ptpClockThreshold	可选。如果 **ptpClockThreshold** 小节不存在，则使用 **ptpClockThreshold** 字段的默认值。小节显示默认的 **ptpClockThreshold** 值。**ptpClockThreshold** 值配置 PTP master 时钟在触发 PTP 事件前的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
ts2phcConf	设置 **ts2phc** 命令的配置。 **leapfile** 是 PTP Operator 容器镜像中当前 leap 秒定义文件的默认路径。 **ts2phc.nmea_serialport** 是连接到 NMEA GPS 时钟源的串行端口设备。配置后，GNSS 接收器可在 **/dev/gnss<id>** 上访问。如果主机有多个 GNSS 接收器，您可以通过枚举以下设备之一来查找正确的设备： • **/sys/class/net/<eth_port>/device/gnss/** • **/sys/class/gnss/gnss<id>/device/**
ts2phcOpts	为 **ts2phc** 命令设置选项。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。
.recommend.profile	指定在 **profile** 部分中定义的 **.recommend.profile** 对象名称。
.recommend.priority	使用 **0** 到 **99** 之间的一个整数值指定 **priority**。大数值的优先级较低，因此优先级 **99** 低于优先级 **10**。如果节点可以根据 **match** 字段中定义的规则与多个配置集匹配，则优先级较高的配置集会应用到该节点。
.recommend.match	使用 **nodeLabel** 或 **nodeName** 值指定 **.recommend.match** 规则。

PtpConfig CR 字段	描述
.recommend.match.nodeLabel	通过 **oc get nodes --show-labels** 命令，使用来自节点对象的 **node.Labels** 的 **key** 设置 **nodeLabel**。例如，**node-role.kubernetes.io/worker**。
.recommend.match.nodeName	使用 **oc get nodes** 命令，将 **nodeName** 设置为来自节点对象的 **node.Name** 值。例如，**compute-1.example.com**。

20.2.6.2. grandmaster 时钟类同步状态参考

下表描述了 PTP grandmaster 时钟(T-GM) gm.ClockClass 状态。时钟类状态根据其准确性和稳定性根据主要参考时间时钟(PRTC)或其他计时来源对 T-GM 时钟进行分类。

holdover 规格是 PTP 时钟可以维护同步的时间，而无需从主时间源接收更新。

表 20.2. T-GM 时钟类状态

时钟类状态	描述
gm.ClockClass 6	T-GM 时钟在 **LOCKED** 模式中连接到 PRTC。例如，PRTC 可以追溯到 GNSS 时间源。
gm.ClockClass 7	T-GM 时钟处于 **HOLDOVER** 模式，在既存的规格内。时钟源可能无法追溯到类别 1 频率源。
gm.ClockClass 140	T-GM 时钟处于 **HOLDOVER** 模式，但仍然可追溯到类别 1 频率源。
gm.ClockClass 248	T-GM 时钟处于 **FREERUN** 模式。

如需更多信息，请参阅 "Phase/time traceability information", ITU-T G.8275.1/Y.1369.1 Recommendations.

20.2.6.3. Intel Westport Channel E810 硬件配置参考

使用这些信息了解如何使用 Intel E810-XXVDA4T 硬件插件 将 E810 网络接口配置为 PTP grandmaster 时钟。硬件固定配置决定了网络接口如何与系统中的其他组件和设备进行交互。E810-XXVDA4T NIC 有四个连接器用于外部 1PPS 信号：SMA1, SMA2, U.FL1, 和 U.FL2。

表 20.3. Intel E810 NIC 硬件连接器配置

硬件固定	推荐的设置	描述
U.FL1	**0 1**	禁用 **U.FL1** 连接器输入。**U.FL1** 连接器是仅用于输出的。
U.FL2	**0 2**	禁用 **U.FL2** 连接器输出。**U.FL2** 连接器是仅限输入的。

硬件固定	推荐的设置	描述
SMA1	**0 1**	禁用 **SMA1** 连接器输入。**SMA1** 连接器是双向的。
SMA2	**0 2**	禁用 **SMA2** 连接器输出。**SMA2** 连接器是双向的。

注意

SMA1 和 U.FL1 连接器共享通道。SMA2 和 U.FL2 连接器共享通道二。

设置 spec.profile.plugins.e810.ublxCmds 参数，以在 PtpConfig 自定义资源(CR) 中配置 GNSS 时钟。这些 ublxCmds 小节各自对应于使用 ubxtool 命令应用到主机 NIC 的配置。例如：

```
ublxCmds:
 - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
   - "-P"
   - "29.20"
   - "-z"
   - "CFG-HW-ANT_CFG_VOLTCTRL,1"
  reportOutput: false
```

下表描述了等效的 ubxtool 命令：

表 20.4. Intel E810 ublxCmds 配置

ubxtool 命令	描述
ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1	启用 atenna voltage 控制。启用在 **UBX-MON-RF** 和 **UBX-INF-NOTICE** 日志消息中报告 antenna 状态。
ubxtool -P 29.20 -e GPS	启用 atenna 接收 GPS 信号。
ubxtool -P 29.20 -d Galileo	配置 atenna 以接收来自 Galileo GPS satellite 的信号。
ubxtool -P 29.20 -d GLONASS	禁用 atenna 从 GLONASS GPS satellite 接收信号。
ubxtool -P 29.20 -d BeiDou	禁用 atenna 从 BeiDou GPS satellite 接收信号。
ubxtool -P 29.20 -d SBAS	禁用 atenna 从 SBAS GPS satellite 接收信号。
ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000	配置 GNSS 接收器调查进程，以提高其初始位置估算。这可能需要 24 小时才能获得最佳结果。

ubxtool 命令	描述
ubxtool -P 29.20 -p MON-HW	对硬件运行单个自动扫描，并报告 NIC 状态和配置设置。

E810 插件实现以下接口：

表 20.5. E810 插件接口

Interface	描述
OnPTPConfigChangeE810	每当您更新 **PtpConfig** CR 时运行。此函数解析插件选项，并根据配置数据将所需的配置应用到网络设备固定。
AfterRunPTPCommandE810	启动 PTP 进程并运行 **gpspipe** PTP 命令后运行。函数处理插件选项并运行 **ubxtool** 命令，将输出存储在特定于插件的数据中。
PopulateHwConfigE810	根据 **PtpConfig** CR 中特定于硬件的数据填充 **NodePtpDevice** CR。

E810 插件有以下 struct 和变量：

表 20.6. E810 插件结构和变量

Struct	描述
E810Opts	代表 E810 插件的选项，包括布尔值标志和网络设备固定映射。
E810UblxCmds	代表带有布尔值标志和命令参数字符串片段的 **ubxtool** 命令的配置。
E810PluginData	包含插件执行期间使用的特定于插件的数据。

20.2.6.4. 双 E810 Westport Channel NIC 配置参考

使用这些信息了解如何使用 Intel E810-XXVDA4T 硬件插件将 E810 网络接口配置为 PTP grandmaster 时钟(T-GM)。

在配置双 NIC 集群主机前，您必须使用 1PPS faceplace 连接将两个 NIC 与 SMA1 电缆连接。

当您配置双 NIC T-GM 时，您需要补补使用 SMA1 连接端口连接 NIC 时发生的 1PPS 信号延迟。电缆长度、基线温度、组件和制造容错等各种因素可能会影响信号延迟。要满足延迟要求，您必须计算用于偏移信号延迟的特定值。

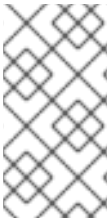
表 20.7. E810 dual-NIC T-GM PtpConfig CR 参考

PtpConfig 字段	描述

PtpConfig 字段	描述
spec.profile.plugins.e810.pins	使用 PTP Operator E810 硬件插件配置 E810 硬件固定。 • 固定 **2 1** 为 NIC 1 上的**SMA1** 启用 **1PPS OUT** 连接。 • 固定 **1 1** 为 NIC 2 上的**SMA1** 启用 **1PPS IN** 连接。
spec.profile.ts2phcConf	使用 **ts2phcConf** 字段为 NIC 1 和 NIC 2 配置参数。为 NIC 2 设置 **ts2phc.master 0**。这会配置来自 1PPS 输入的 NIC 2 的计时源，而不是 GNSS。为 NIC 2 配置 **ts2phc.extts_correction** 值，以补偿您所使用的特定 SMA 电缆和电缆长度的延迟。您配置的值取决于您的特定测量和 SMA1 电缆长度。
spec.profile.ptp4lConf	将 **boundary_clock_jbod** 的值设置为 1，以启用对多个 NIC 的支持。

20.2.7. 将 linuxptp 服务配置为边界时钟

您可以通过创建 PtpConfig 自定义资源(CR)对象将 linuxptp 服务（ptp4l、phc2sys）配置为边界时钟。



注意

使用 PtpConfig CR 示例，将 linuxptp 服务配置为特定硬件和环境的边界时钟。这个示例 CR 没有配置 PTP 快速事件。要配置 PTP 快速事件，请为 ptp4lOpts、ptp4lConf 和 ptpClockThreshold 设置适当的值。ptpClockThreshold 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**
- **安装 PTP Operator。**

流程

1. **创建以下 PtpConfig CR，然后在 boundaries-clock-ptp-config.yaml 文件中保存 YAML。**

   PTP 边界时钟配置示例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: boundary-clock
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
```

```
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    ptp4lConf: |
      # The interface name is hardware-specific
      [$iface_slave]
      masterOnly 0
      [$iface_master_1]
      masterOnly 1
      [$iface_master_2]
      masterOnly 1
      [$iface_master_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      slaveOnly 0
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 248
      clockAccuracy 0xFE
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
      dscp_event 0
      dscp_general 0
      dataset_comparison G.8275.x
      G.8275.defaultDS.localPriority 128
      #
      # Port Data Set
      #
      logAnnounceInterval -3
      logSyncInterval -4
      logMinDelayReqInterval -4
      logMinPdelayReqInterval -4
      announceReceiptTimeout 3
      syncReceiptTimeout 0
      delayAsymmetry 0
      fault_reset_interval -4
      neighborPropDelayThresh 20000000
      masterOnly 0
      G.8275.portDS.localPriority 128
      #
      # Run time options
      #
      assume_two_step 0
      logging_level 6
      path_trace_enabled 0
      follow_up_info 0
      hybrid_e2e 0
      inhibit_multicast_service 0
```

```
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
```

```
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
recommend:
  - profile: boundary-clock
    priority: 4
    match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"
```

表 20.8. PTP 边界时钟 CR 配置选项

CR 字段	描述
name	**PtpConfig** CR 的名称。
配置集	指定包括一个或多个 **profile** 的数组。
name	指定唯一标识配置集对象的配置集对象的名称。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。
ptp4lConf	指定启动 **ptp4l** 作为边界时钟所需的配置。例如，**ens1f0** 同步来自 Pumaster 时钟，**ens1f3** 同步连接的设备。
<interface_1>	接收同步时钟的接口。
<interface_2>	发送同步时钟的接口。
tx_timestamp_timeout	对于 Intel Columbiaville 800 系列 NIC，将 **tx_timestamp_timeout** 设置为 **50**。
boundary_clock_jbod	对于 Intel Columbiaville 800 系列 NIC，请确保 **boundary_clock_jbod** 设置为 **0**。对于 Intel Fortville X710 系列 NIC，请确保 **boundary_clock_jbod** 设置为 **1**。
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。
ptpSchedulingPolicy	ptp4l 和 phc2sys 进程的调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，用于为 **ptp4l** 和 **phc2sys** 进程设置 FIFO 优先级的整数值（1 到 65）。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。

CR 字段	描述
ptpClockThreshold	可选。如果没有 **ptpClockThreshold**，用于 **ptpClockThreshold** 字段的默认值。**ptpClockThreshold** 配置在触发 PTP 时间前，PTP master 时钟已断开连接的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。
.recommend.profile	指定在 **profile** 部分定义的 **.recommend.profile** 对象名称。
.recommend.priority	使用 **0** 到 **99** 之间的一个整数值指定 **priority**。大数值的优先级较低，因此优先级 **99** 低于优先级 **10**。如果节点可以根据 **match** 字段中定义的规则与多个配置集匹配，则优先级较高的配置集会应用到该节点。
.recommend.match	使用 **nodeLabel** 或 **nodeName** 值指定 **.recommend.match** 规则。
.recommend.match.nodeLabel	通过 **oc get nodes --show-labels** 命令，使用来自节点对象的 **node.Labels** 的 **key** 设置 **nodeLabel**。例如，**node-role.kubernetes.io/worker**。
.recommend.match.nodeName	使用 **oc get nodes** 命令，将 **nodeName** 设置为来自节点对象的 **node.Name** 值。例如，**compute-1.example.com**。

2. **运行以下命令来创建 CR：**

```
$ oc create -f boundary-clock-ptp-config.yaml
```

验证

1. **检查 PtpConfig 配置集是否已应用到节点。**
   a. **运行以下命令，获取 openshift-ptp 命名空间中的 pod 列表：**

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                     READY   STATUS    RESTARTS   AGE   IP           NODE
linuxptp-daemon-4xkbb    1/1     Running   0          43m   10.1.196.24
compute-0.example.com
linuxptp-daemon-tdspf    1/1     Running   0          43m   10.1.196.25   compute-
```

```
1.example.com
ptp-operator-657bbb64c8-2f8sj   1/1     Running   0          43m   10.129.0.61
control-plane-1.example.com
```

b. 检查配置集是否正确。检查与 PtpConfig 配置集中指定的节点对应的 linuxptp 守护进程的日志。运行以下命令:

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

输出示例

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

其他资源

- 为 PTP 硬件配置 FIFO 优先级调度
- 配置 PTP 快速事件通知发布程序

20.2.7.1. 将 linuxptp 服务配置为双 NIC 硬件的边界时钟

您可以通过为每个 NIC 创建一个 PtpConfig 自定义资源(CR)对象，将 linuxptp 服务（ptp4l、phc2sys）配置为双 NIC 硬件的边界时钟。

双 NIC 硬件允许您将每个 NIC 连接到相同的上游领导时钟，并将每个 NIC 的 ptp4l 实例连接给下游时钟。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建两个单独的 PtpConfig CR，每个 NIC 使用 "Configuring linuxptp 服务作为边界时钟"中的引用 CR，作为每个 CR 的基础。例如：

   a. 创建 boundary-clock-ptp-config-nic1.yaml，为 phc2sysOpts 指定值：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic1
  namespace: openshift-ptp
spec:
```

```
  profile:
  - name: "profile1"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: | 1
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    ...
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
```

1 指定所需的接口来启动 ptp4l 作为一个边境时钟。例如，ens5f0 从 grandmaster 时钟同步，ens5f1 同步连接的设备。

2 所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

b. 创建 boundary-clock-ptp-config-nic2.yaml，删除 phc2sysOpts 字段，以完全禁用第二个 NIC 的 phc2sys 服务：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: | 1
      [ens7f1]
      masterOnly 1
      [ens7f0]
      masterOnly 0
...
```

1 在第二个 NIC上 指定所需的接口来启动 ptp4l 作为一个边境时钟。

注意

您必须从第二个 PtpConfig CR 中完全删除phc2sysOpts 字段，以禁用第二个 NIC 上的 phc2sys 服务。

2. 运行以下命令来创建双 NIC PtpConfig CR：

a. 创建 CR 来为第一个 NIC 配置 PTP：

```
$ oc create -f boundary-clock-ptp-config-nic1.yaml
```

b. 创建 CR 来为第二个 NIC 配置 PTP：

```
$ oc create -f boundary-clock-ptp-config-nic2.yaml
```

验证

- 检查 PTP Operator 是否为两个 NIC 应用了 PtpConfig CR。检查与安装了双 NIC 硬件的节点对应的 linuxptp 守护进程的日志。例如，运行以下命令：

```
$ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
```

输出示例

```
ptp4l[80828.335]: [ptp4l.1.config] master offset          5 s2 freq   -5727 path delay       519
ptp4l[80828.343]: [ptp4l.0.config] master offset         -5 s2 freq  -10607 path delay
533
phc2sys[80828.390]: [ptp4l.0.config] CLOCK_REALTIME phc offset         1 s2 freq  -
87239 delay    539
```

20.2.7.2. 将 linuxptp 配置为双 NIC Intel E810 PTP 边界时钟的高可用性系统时钟

您可以将 linuxptp 服务 ptp4l 和 phc2sys 配置为双 PTP 边界时钟 (T-BC) 的高可用性 (HA) 系统时钟。

高可用性系统时钟使用来自双 NIC Intel E810 Salem 频道硬件的多个时间源，配置为两个边界时钟。两个边界时钟实例参与 HA 设置，每个设置都有自己的配置 profile。您可以将每个 NIC 连接到相同的上游领导时钟，每个 NIC 为下游时钟提供单独的 ptp4l 实例。

创建两个 PtpConfig 自定义资源 (CR) 对象，将 NIC 配置为 T-BC 和第三个PtpConfig CR，以配置两个 NIC 之间的高可用性。

重要

当您创建 PtpConfig CR 时，请确保 phc2sysOpts 字段是一个空字符串，以防止在这两个配置集上设置 phc2sys 进程。

第三个 PtpConfig CR 配置高度可用的系统时钟服务。CR 将ptp4lOpts 字段设置为空字符串，以防止 ptp4l 进程运行。CR 在 spec.profile.ptpSettings.haProfiles 键下添加 ptp4l 配置的配置集，并将这些配置集的内核套接字路径传递给 phc2sys 服务。当出现 ptp4l 失败时，phc2sys 服务将切换到备份 ptp4l 配置。当主配置集再次激活时，phc2sys 服务将恢复到原始状态。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 PTP Operator。
- 使用 Intel E810 Salem 频道双 NIC 配置集群节点。

流程

1. 创建两个单独的 PtpConfig CR，每个 NIC 使用"将 linuxptp 服务配置为双 NIC 硬件边界时钟"中的 CR 作为每个 CR 的引用。
   a. 创建 ha-ptp-config-nic1.yaml 文件，为 phc2sysOpts 字段指定一个空字符串。例如：

```
apiVersion: ptp.openshift.io/v1
```

```
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
  - name: "ha-ptp-config-profile1"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: | 1
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    #...
    phc2sysOpts: "" 2
```

1 指定所需的接口来启动 ptp4l 作为一个边境时钟。例如，ens5f0 从 grandmaster 时钟同步，ens5f1 同步连接的设备。

2 使用空字符串设置 phc2sysOpts。这些值从配置高可用性的 PtpConfig CR 的 spec.profile.ptpSettings.haProfiles 字段填充。

b. 运行以下命令，为 NIC 1 应用 PtpConfig CR：

```
$ oc create -f ha-ptp-config-nic1.yaml
```

c. 创建 ha-ptp-config-nic2.yaml 文件，为 phc2sysOpts 字段指定一个空字符串。例如：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "ha-ptp-config-profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
      [ens7f1]
      masterOnly 1
      [ens7f0]
      masterOnly 0
    #...
    phc2sysOpts: ""
```

d. 运行以下命令，为 NIC 2 应用 PtpConfig CR：

```
$ oc create -f ha-ptp-config-nic2.yaml
```

2. 创建配置 HA 系统时钟的 PtpConfig CR。例如：

a. 创建 ptp-config-for-ha.yaml 文件：

```
apiVersion: ptp.openshift.io/v1
```

```
kind: PtpConfig
metadata:
  name: boundary-ha
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: "boundary-ha"
      ptp4lOpts: " "
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
        haProfiles: "ha-ptp-config-nic1,ha-ptp-config-nic2"
  recommend:
    - profile: "boundary-ha"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

重要

在配置单个 NIC 的 PtpConfig CR 前，不要应用高可用性PtpConfig CR。

b. 运行以下命令来应用 HA PtpConfig CR：

```
$ oc create -f ptp-config-for-ha.yaml
```

验证

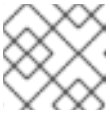
- 验证 PTP Operator 是否已正确应用 PtpConfig CR。执行以下步骤：

a. 运行以下命令，获取 openshift-ptp 命名空间中的 pod 列表：

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                    READY   STATUS    RESTARTS   AGE   IP           NODE
linuxptp-daemon-4xkrb   1/1     Running   0          43m   10.1.196.24  compute-
0.example.com
ptp-operator-657bbq64c8-2f8sj   1/1     Running   0          43m   10.129.0.61
control-plane-1.example.com
```



注意

应该只有一个 linuxptp-daemon pod。

b. 运行以下命令，检查配置集是否正确。检查与 PtpConfig 配置集中指定的节点对应的 linuxptp 守护进程的日志。

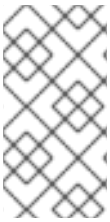
```
$ oc logs linuxptp-daemon-4xkrb -n openshift-ptp -c linuxptp-daemon-container
```

输出示例

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: ha-ptp-config-profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

20.2.8. 将 linuxptp 服务配置为常规时钟

您可以通过创建 PtpConfig 自定义资源(CR)对象将 linuxptp 服务（ptp4l、phc2sys）配置为常规时钟。



注意

使用 PtpConfig CR 示例，将 linuxptp 服务配置为特定硬件和环境的普通时钟。这个示例 CR 没有配置 PTP 快速事件。要配置 PTP 快速事件，请为 ptp4lOpts、ptp4lConf 和 ptpClockThreshold 设置适当的值。只有在启用事件时才需要 ptpClockThreshold。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建以下 PtpConfig CR，然后在 ordinary-clock-ptp-config.yaml 文件中保存 YAML。

   PTP 普通时钟配置示例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: ordinary-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
```

```
ptpSettings:
  logReduce: "true"
ptp4lConf: |
  [global]
  #
  # Default Data Set
  #
  twoStepFlag 1
  slaveOnly 1
  priority1 128
  priority2 128
  domainNumber 24
  #utc_offset 37
  clockClass 255
  clockAccuracy 0xFE
  offsetScaledLogVariance 0xFFFF
  free_running 0
  freq_est_interval 1
  dscp_event 0
  dscp_general 0
  dataset_comparison G.8275.x
  G.8275.defaultDS.localPriority 128
  #
  # Port Data Set
  #
  logAnnounceInterval -3
  logSyncInterval -4
  logMinDelayReqInterval -4
  logMinPdelayReqInterval -4
  announceReceiptTimeout 3
  syncReceiptTimeout 0
  delayAsymmetry 0
  fault_reset_interval -4
  neighborPropDelayThresh 20000000
  masterOnly 0
  G.8275.portDS.localPriority 128
  #
  # Run time options
  #
  assume_two_step 0
  logging_level 6
  path_trace_enabled 0
  follow_up_info 0
  hybrid_e2e 0
  inhibit_multicast_service 0
  net_sync_monitor 0
  tc_spanning_tree 0
  tx_timestamp_timeout 50
  unicast_listen 0
  unicast_master_table 0
  unicast_req_duration 3600
  use_syslog 1
  verbose 0
  summary_interval 0
  kernel_leap 1
  check_fup_sync 0
```

```
      clock_class_threshold 7
      #
      # Servo Options
      #
      pi_proportional_const 0.0
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: ordinary-clock
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

表 20.9. PTP 普通时钟 CR 配置选项

CR 字段	描述
name	**PtpConfig** CR 的名称。
配置集	指定包括一个或多个 **profile** 的数组。每个配置集的名称都需要是唯一的。
interface	指定 **ptp4l** 服务要使用的网络接口，如 **ens787f1**。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项，例如 **-2** 来选择 IEEE 802.3 网络传输。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。附加 **--summary_interval -4** 来对此接口使用 PTP 快速事件。
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。对于 Intel Columbiaville 800 Series NIC，将 **phc2sysOpts** 选项设置为 **-a -r -m -n 24 -N 8 -R 16**.**-m** 将消息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。
ptp4lConf	指定一个字符串，其中包含要替换默认的 **/etc/ptp4l.conf** 文件的配置。要使用默认配置，请将字段留空。
tx_timestamp_timeout	对于 Intel Columbiaville 800 系列 NIC，将 **tx_timestamp_timeout** 设置为 **50**。
boundary_clock_jbod	对于 Intel Columbiaville 800 系列 NIC，将 **boundary_clock_jbod** 设置为 **0**。
ptpSchedulingPolicy	**ptp4l** 和 **phc2sys** 进程的调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，用于为 **ptp4l** 和 **phc2sys** 进程设置 FIFO 优先级的整数值（1 到 65）。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。
ptpClockThreshold	可选。如果没有 **ptpClockThreshold**，用于 **ptpClockThreshold** 字段的默认值。**ptpClockThreshold** 配置在触发 PTP 时间前，PTP master 时钟已断开连接的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。

CR 字段	描述
.recommend.profile	指定在 profile 部分定义的 .recommend.profile 对象名称。
.recommend.priority	对于普通时钟，将 .recommend.priority 设置为 0。
.recommend.match	使用 nodeLabel 或 nodeName 值指定 .recommend.match 规则。
.recommend.match.nodeLabel	通过 oc get nodes --show-labels 命令，使用来自节点对象的 node.Labels 的 key 设置 nodeLabel。例如，node-role.kubernetes.io/worker。
.recommend.match.nodeName	使用 oc get nodes 命令，将 nodeName 设置为来自节点对象的 node.Name 值。例如，compute-1.example.com。

2. 运行以下命令来创建 PtpConfig CR：

```
$ oc create -f ordinary-clock-ptp-config.yaml
```

验证

1. 检查 PtpConfig 配置集是否已应用到节点。

   a. 运行以下命令，获取 openshift-ptp 命名空间中的 pod 列表：

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   输出示例

   ```
   NAME                            READY   STATUS    RESTARTS   AGE   IP           NODE
   linuxptp-daemon-4xkbb           1/1     Running   0          43m   10.1.196.24
   compute-0.example.com
   linuxptp-daemon-tdspf           1/1     Running   0          43m   10.1.196.25     compute-
   1.example.com
   ptp-operator-657bbb64c8-2f8sj   1/1     Running   0          43m   10.129.0.61
   control-plane-1.example.com
   ```

   b. 检查配置集是否正确。检查与 PtpConfig 配置集中指定的节点对应的 linuxptp 守护进程的日志。运行以下命令:

   ```
   $ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
   ```

   输出示例

   ```
   I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
   I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
   ```

```
I1115 09:41:17.117607 4143292 daemon.go:110] ----------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2 -s
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ----------------------------------
```

其他资源

- 为 PTP 硬件配置 FIFO 优先级调度
- 配置 PTP 快速事件通知发布程序

20.2.8.1. Intel Columbiaville E800 series NIC 作为 PTP 常规时钟参考

下表描述了您必须对引用 PTP 配置所做的更改，以使用 Intel Columbiaville E800 系列 NIC 作为普通时钟。在应用到集群的 PtpConfig 自定义资源(CR)中进行更改。

表 20.10. Intel Columbiaville NIC 的推荐 PTP 设置

PTP 配置	推荐的设置
phc2sysOpts	-a -r -m -n 24 -N 8 -R 16
tx_timestamp_timeout	50
boundary_clock_jbod	0

注意

对于 phc2sysOpts，-m 会将信息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。

其他资源

- 有关将 linuxptp 服务配置为具有 PTP 快速事件的普通时钟的完整示例 CR，请参阅将 linuxptp 服务配置为普通时钟。

20.2.9. 为 PTP 硬件配置 FIFO 优先级调度

在需要低延迟性能的电信或其他部署类型中，PTP 守护进程线程在受限的 CPU 占用空间以及剩余的基础架构组件一起运行。默认情况下，PTP 线程使用 SCHED_OTHER 策略运行。在高负载下，这些线程可能没有获得无错操作所需的调度延迟。

要缓解潜在的调度延迟错误，您可以将 PTP Operator linuxptp 服务配置为允许线程使用 SCHED_FIFO 策略运行。如果为 PtpConfig CR 设置了 SCHED_FIFO，则 ptp4l 和 phc2sys 将在 chrt 的父容器中运行，且由 PtpConfig CR 的ptpSchedulingPriority 字段设置。

注意

设置 ptpSchedulingPolicy 是可选的，只有在遇到延迟错误时才需要。

流程

1. 编辑 PtpConfig CR 配置集：

```
$ oc edit PtpConfig -n openshift-ptp
```

2. 更改 ptpSchedulingPolicy 和 ptpSchedulingPriority 字段：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSchedulingPolicy: SCHED_FIFO (1)
    ptpSchedulingPriority: 10 (2)
```

(1) ptp4l 和 phc2sys 进程的调度策略。在支持 FIFO 调度的系统上使用 SCHED_FIFO。

(2) 必需。设置整数值 1-65，用于为 ptp4l 和 phc2sys 进程配置 FIFO 优先级。

3. 保存并退出，以将更改应用到 PtpConfig CR。

验证

1. 获取 linuxptp-daemon pod 的名称以及应用 PtpConfig CR 的对应节点：

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                     READY   STATUS    RESTARTS   AGE    IP           NODE
linuxptp-daemon-gmv2n    3/3     Running   0          1d17h  10.1.196.24  compute-
0.example.com
linuxptp-daemon-lgm55    3/3     Running   0          1d17h  10.1.196.25  compute-
1.example.com
ptp-operator-3r4dcvf7f4-zndk7  1/1   Running   0          1d7h   10.129.0.61  control-
plane-1.example.com
```

2. 检查 ptp4l 进程是否使用更新的 chrt FIFO 运行：

```
$ oc -n openshift-ptp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep
chrt
```

输出示例

```
I1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ptp4l -f
/var/run/ptp4l.0.config -2  --summary_interval -4 -m
```

20.2.10. 为 linuxptp 服务配置日志过滤

linuxptp 守护进程生成可用于调试目的的日志。在具有有限存储容量的电信或其他部署类型中，这些日志可以添加到存储要求中。

要减少数量日志消息，您可以配置 PtpConfig 自定义资源 (CR) 来排除报告master offset 值的日志消息。master offset 日志消息以纳秒为单位报告当前节点时钟和 master 时钟之间的区别。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**
- **安装 PTP Operator。**

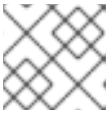
流程

1. **编辑 PtpConfig CR：**

```
$ oc edit PtpConfig -n openshift-ptp
```

2. **在 spec.profile 中，添加 ptpSettings.logReduce 规格，并将值设为 true ：**

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSettings:
      logReduce: "true"
```



注意

为了进行调试，您可以将此规格恢复到 False，使其包含 master 偏移消息。

3. **保存并退出，以将更改应用到 PtpConfig CR。**

验证

1. **获取 linuxptp-daemon pod 的名称以及应用 PtpConfig CR 的对应节点：**

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                    READY   STATUS    RESTARTS   AGE    IP           NODE
linuxptp-daemon-gmv2n   3/3     Running   0          1d17h  10.1.196.24  compute-
```

```
0.example.com
linuxptp-daemon-lgm55          3/3     Running   0          1d17h   10.1.196.25   compute-
1.example.com
ptp-operator-3r4dcvf7f4-zndk7  1/1     Running   0          1d7h    10.129.0.61   control-
plane-1.example.com
```

2. 运行以下命令，验证 master 偏移信息是否不包括在日志中：

```
$ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container |
grep "master offset" 1
```

1 <linux_daemon_container> 是 linuxptp-daemon pod 的名称，如 linuxptp-daemon-gmv2n。

当您配置 logReduce 规格时，这个命令会在 linuxptp 守护进程日志中报告任何 master offset 实例。

20.2.11. 常见 PTP Operator 故障排除

通过执行以下步骤排除 PTP Operator 中的常见问题。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 使用支持 PTP 的主机在裸机集群中安装 PTP Operator。

流程

1. 检查集群中为配置的节点成功部署了 Operator 和操作对象。

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                            READY   STATUS    RESTARTS   AGE     IP           NODE
linuxptp-daemon-lmvgn           3/3     Running   0          4d17h   10.1.196.24   compute-
0.example.com
linuxptp-daemon-qhfg7           3/3     Running   0          4d17h   10.1.196.25   compute-
1.example.com
ptp-operator-6b8dcbf7f4-zndk7   1/1     Running   0          5d7h    10.129.0.61   control-
plane-1.example.com
```

注意

当启用 PTP fast 事件总线时，就绪的 linuxptp-daemon pod 的数量是 3/3。如果没有启用 PTP fast 事件总线，则会显示 2/2。

2. 检查集群中是否已找到支持的硬件。

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

输出示例

```
NAME                         AGE
control-plane-0.example.com   10d
control-plane-1.example.com   10d
compute-0.example.com         10d
compute-1.example.com         10d
compute-2.example.com         10d
```

3. 检查节点的可用 PTP 网络接口：

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

其中：

<node_name>
指定您要查询的节点，例如 compute-0.example.com。

输出示例

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2021-09-14T16:52:33Z"
  generation: 1
  name: compute-0.example.com
  namespace: openshift-ptp
  resourceVersion: "177400"
  uid: 30413db0-4d8d-46da-9bef-737bacd548fd
spec: {}
status:
  devices:
  - name: eno1
  - name: eno2
  - name: eno3
  - name: eno4
  - name: enp5s0f0
  - name: enp5s0f1
```

4. 通过访问对应节点的 linuxptp-daemon pod，检查 PTP 接口是否已与主时钟成功同步。

a. 运行以下命令来获取 linuxptp-daemon pod 的名称以及您要排除故障的对应节点：

```
$ oc get pods -n openshift-ptp -o wide
```

输出示例

```
NAME                     READY   STATUS    RESTARTS   AGE    IP           NODE
linuxptp-daemon-lmvgn    3/3     Running   0          4d17h  10.1.196.24
compute-0.example.com
```

```
linuxptp-daemon-qhfg7          3/3     Running   0          4d17h   10.1.196.25   compute-
1.example.com
ptp-operator-6b8dcbf7f4-zndk7  1/1     Running   0          5d7h    10.129.0.61
control-plane-1.example.com
```

b. 在远程 shell 到所需的 linuxptp-daemon 容器：

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
```

其中：

<linux_daemon_container>

您要诊断的容器，如 linuxptp-daemon-lmvgn。

c. 在与 linuxptp-daemon 容器的远程 shell 连接中，使用 PTP Management Client (pmc) 工具诊断网络接口。运行以下 pmc 命令，以检查 PTP 设备的同步状态，如ptp4l。

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

当节点成功同步到主时钟时的输出示例

```
sending: GET PORT_DATA_SET
   40a6b7.fffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
      portIdentity            40a6b7.fffe.166ef0-1
      portState               SLAVE
      logMinDelayReqInterval  -4
      peerMeanPathDelay       0
      logAnnounceInterval     -3
      announceReceiptTimeout  3
      logSyncInterval         -4
      delayMechanism          1
      logMinPdelayReqInterval -4
      versionNumber           2
```

5. 对于 GNSS-sourced grandmaster 时钟，运行以下命令来验证 in-tree NIC ice 驱动程序是否正确，例如：

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-74m2g
ethtool -i ens7f0
```

输出示例

```
driver: ice
version: 5.14.0-356.bz2232515.el9.x86_64
firmware-version: 4.20 0x8001778b 1.3346.0
```

6. 对于 GNSS-sourced grandmaster 时钟，请验证 linuxptp-daemon 容器是否从 GNSS antenna 接收信号。如果容器没有收到 GNSS 信号，则不会填充 /dev/gnss0 文件。要验证，请运行以下命令：

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-jnz6r cat
/dev/gnss0
```

输出示例

```
$GNRMC,125223.00,A,4233.24463,N,07126.64561,W,0.000,,300823,,,A,V*0A
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGGA,125223.00,4233.24463,N,07126.64561,W,1,12,99.99,98.6,M,-33.1,M,,*7E
$GNGSA,A,3,25,17,19,11,12,06,05,04,09,20,,,99.99,99.99,99.99,1*37
$GPGSV,3,1,10,04,12,039,41,05,31,222,46,06,50,064,48,09,28,064,42,1*62
```

20.2.12. 在 Intel 800 系列 NIC 中获取 CGU 的 DPLL 固件版本

您可以通过打开 debug shell 到集群节点并查询 NIC 硬件，在 Intel 800 系列 NIC 中获取 Clock Generation Unit (CGU) 的数字签名循环 (DPLL) 固件版本。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已在集群主机中安装了 Intel 800 系列 NIC。
- 您已在带有支持 PTP 的主机的裸机集群中安装 PTP Operator。

流程

1. 运行以下命令来启动 debug pod：

```
$ oc debug node/<node_name>
```

其中：

<node_name>
是安装 Intel 800 系列 NIC 的节点。

2. 使用 devlink 工具以及安装 NIC 的总线和设备名称，检查 NIC 中的 CGU 固件版本。例如，运行以下命令：

```
sh-4.4# devlink dev info <bus_name>/<device_name> | grep cgu
```

其中：

<bus_name>
是安装 NIC 的总线。例如，pci。

<device_name>
是 NIC 设备名称。例如，0000:51:00.0。

输出示例

```
cgu.id 36 1
fw.cgu 8032.16973825.6021 2
```

1 CGU 硬件修订号

2 在 CGU 中运行的 DPLL 固件版本，DPLL 固件版本为 6201，DPLL 模型是 8032。字符串 16973825 是 DPLL 固件版本的二进制版本的简写形式 (1.3.0.1)。

注意

固件版本的每个版本号部分都包括了前导和 3 个八位字节位。数字 16973825 的二进制格式是 0001 0000 0011 0000 0000 0000 0001。使用二进制值来解码固件版本。例如：

表 20.11. DPLL 固件版本

二进制部分	十进制值
0001	1
0000 0011	3
0000 0000	0
0000 0001	1

20.2.13. 收集 PTP Operator 数据

您可以使用 oc adm must-gather 命令收集有关集群的信息，包括与 PTP Operator 关联的功能和对象。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。
- 已安装 PTP Operator。

流程

- 要使用 must-gather 来收集 PTP Operator 数据，您必须指定 PTP Operatormust-gather 镜像。

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel8:v4.16
```

20.3. 使用 PTP 硬件快速事件通知框架

虚拟 RAN (vRAN) 等云原生应用需要访问对整个网络运行至关重要的硬件计时事件通知。PTP 时钟同步错误可能会对低延迟应用程序的性能和可靠性造成负面影响，例如：在一个分布式单元 (DU) 中运行的 vRAN 应用程序。

20.3.1. 关于 PTP 和时钟同步错误事件

丢失 PTP 同步是 RAN 网络的一个关键错误。如果在节点上丢失同步，则可能会关闭无线广播，并且网络 Over the Air (OTA) 流量可能会转移到无线网络中的另一个节点。快速事件通知允许集群节点与 DU 中运行的 vRAN 应用程序通信 PTP 时钟同步状态，从而缓解工作负载错误。

事件通知可用于在同一 DU 节点上运行的 vRAN 应用。发布/订阅 REST API 将事件通知传递到消息传递总线。发布-订阅消息传递或发布-订阅消息传递是服务通信架构的异步服务，通过服务通信架构，所有订阅者会立即收到发布到某一主题的消息。

PTP Operator 为每个支持 PTP 的网络接口生成快速事件通知。您可以通过 HTTP 或 Advanced Message Queuing Protocol (AMQP) 消息总线使用 **cloud-event-proxy** sidecar 容器来访问事件。

注意

PTP 快速事件通知可用于配置为使用 PTP 普通时钟、PTP grandmaster 时钟或 PTP 边界时钟。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

20.3.2. 关于 PTP 快速事件通知框架

使用 Precision Time Protocol (PTP) 快速事件通知框架，将集群应用程序订阅到裸机集群节点的 PTP 事件。

注意

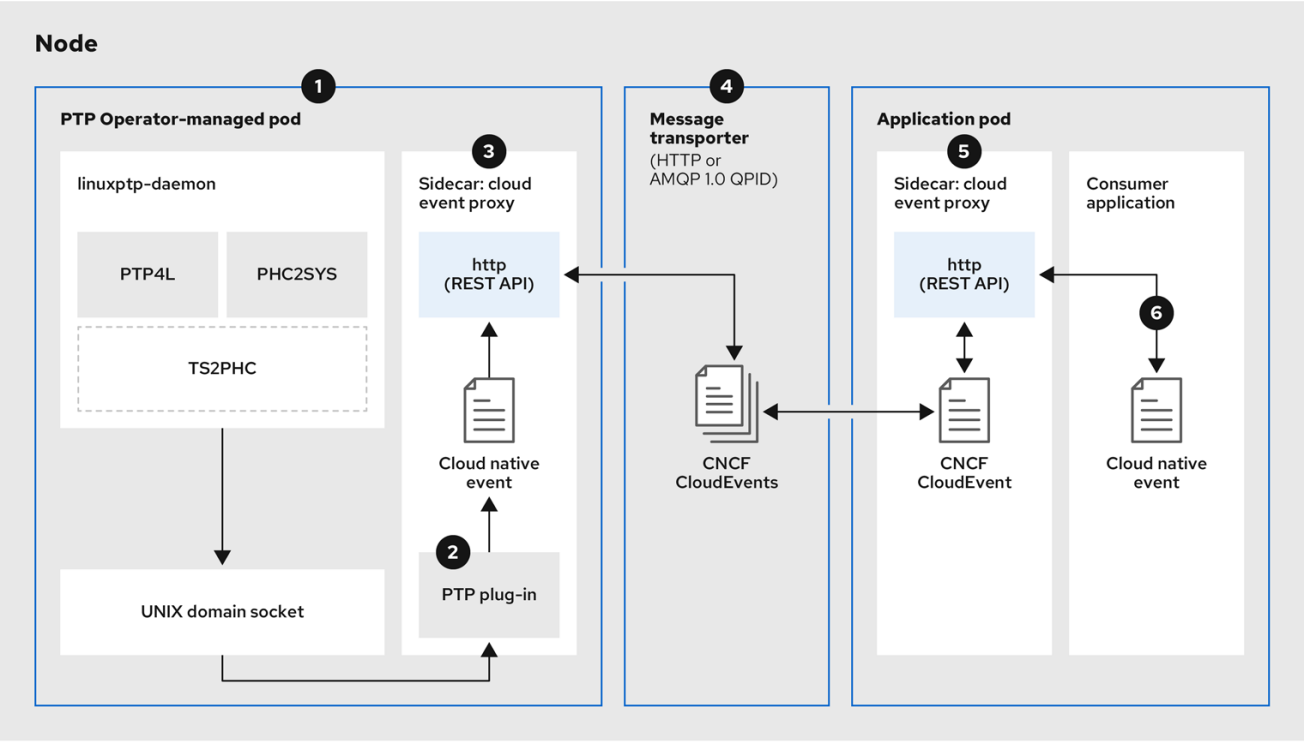
快速事件通知框架使用 REST API 进行通信。REST API 基于 *O-RAN O-Cloud Notification API Specification for Event Consumers 3.0*，它包括在 O-RAN ALLIANCE Specifications 中。

框架由发布者、订阅者和 AMQ 或 HTTP 消息传递协议组成，用于处理发布者和订阅者应用程序之间的通信。应用程序以 sidecar 模式运行 **cloud-event-proxy** 容器，以订阅 PTP 事件。**cloud-event-proxy** sidecar 容器可以访问与主应用程序容器相同的资源，而无需使用主应用程序的任何资源，且没有大量延迟。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

图 20.4. PTP 快速事件概述



1 事件在集群主机上生成

PTP Operator 管理的 pod 中的 **linuxptp-daemon** 作为 Kubernetes **DaemonSet** 运行，并管理各种 **linuxptp** 进程 (**ptp4l**、**phc2sys**，以及可选的用于 grandmaster 时钟 **ts2phc**)。**linuxptp-daemon** 将事件传递给 UNIX 域套接字。

2 事件传递给 cloud-event-proxy sidecar

PTP 插件从 UNIX 域套接字读取事件，并将其传递给 PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar。**cloud-event-proxy** 将 Kubernetes 基础架构的事件提供给具有低延迟的 Cloud-Native Network Function (CNF)。

3 事件是持久的

PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar 处理事件，并使用 REST API 发布云原生事件。

4 消息已传输

消息传输程序通过 HTTP 或 AMQP 1.0 QPID 将事件传送到应用程序 pod 中的 **cloud-event-proxy** sidecar。

5 来自 REST API 的事件

Application pod 中的 **cloud-event-proxy** sidecar 处理事件并使用 REST API 使其可用。

6 消费者应用程序请求订阅并接收订阅的事件

消费者应用程序向应用程序 pod 中的 **cloud-event-proxy** sidecar 发送 API 请求，以创建 PTP 事件订阅。**cloud-event-proxy** sidecar 为订阅中指定的资源创建一个 AMQ 或 HTTP 消息传递监听程序协议。

应用程序 pod 中的 cloud-event-proxy sidecar 接收来自 PTP Operator 管理的 pod 的事件，取消封装云事件对象以检索数据，并将事件发布到消费者应用程序。消费者应用程序侦听资源限定符中指定的地址，并接收和处理 PTP 事件。

20.3.3. 配置 PTP 快速事件通知发布程序

要为集群中的网络接口启动使用 PTP fast 事件通知，您必须在 PTP Operator PtpOperatorConfig 自定义资源 (CR) 中启用快速事件发布程序，并在您创建的 PtpConfig CR 中配置 ptpClockThreshold 值。

先决条件

- 已安装 OpenShift Container Platform CLI (oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 已安装 PTP Operator。

流程

1. 修改默认 PTP Operator 配置以启用 PTP 快速事件。
   a. 在 ptp-operatorconfig.yaml 文件中保存以下 YAML：

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true (1)
```

(1) 将 enableEventPublisher 设置为 true 以启用 PTP 快速事件通知。

注意

在 OpenShift Container Platform 4.13 或更高版本中，当将 HTTP 传输用于 PTP 事件时，您不需要在 PtpOperatorConfig 资源中设置 spec.ptpEventConfig.transportHost 字段。仅在 PTP 事件中使用 AMQP 传输时设置 transportHost。

   a. 更新 PtpOperatorConfig CR：

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. 为 PTP 启用接口创建 PtpConfig 自定义资源(CR)，并设置 ptpClockThreshold 和 ptp4lOpts 所需的值。以下 YAML 演示了您必须在 PtpConfig CR 中设置的必要值：

```
spec:
  profile:
  - name: "profile1"
```

```
interface: "enp5s0f0"
ptp4lOpts: "-2 -s --summary_interval -4" 1
phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
ptp4lConf: "" 3
ptpClockThreshold: 4
  holdOverTimeout: 5
  maxOffsetThreshold: 100
  minOffsetThreshold: -100
```

1. 附加 --summary_interval -4 以使用 PTP 快速事件。
2. 所需的 phc2sysOpts 值。-m 将消息输出到 stdout。linuxptp-daemon DaemonSet 解析日志并生成 Prometheus 指标。
3. 指定一个字符串，其中包含要替换默认的 /etc/ptp4l.conf 文件的配置。要使用默认配置，请将字段留空。
4. 可选。如果 ptpClockThreshold 小节不存在，则默认值用于 ptpClockThreshold 字段。小节显示默认的 ptpClockThreshold 值。ptpClockThreshold 值配置 PTP master 时钟在触发 PTP 事件前的时长。holdOverTimeout 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 FREERUN 前的时间值（以秒为单位）。maxOffsetThreshold 和 minOffsetThreshold 设置以纳秒为单位，它们与 CLOCK_REALTIME (phc2sys) 或 master 偏移 (ptp4l) 的值进行比较。当 ptp4l 或 phc2sys 偏移值超出这个范围时，PTP 时钟状态被设置为 FREERUN。当偏移值在这个范围内时，PTP 时钟状态被设置为 LOCKED。

其他资源

- 有关将 linuxptp 服务配置为具有 PTP 快速事件的普通时钟的完整示例 CR，请参阅将 linuxptp 服务配置为普通时钟。

20.3.4. 迁移消费者应用程序，以使用 PTP 或裸机事件的 HTTP 传输

如果您之前部署了 PTP 或裸机事件消费者应用程序，您需要更新应用程序以使用 HTTP 消息传输。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已将 PTP Operator 或 Bare Metal Event Relay 更新至使用 HTTP 传输的版本 4.13+。

流程

1. 更新您的事件消费者应用以使用 HTTP 传输。为云事件 sidecar 部署设置 http-event-publishers 变量。
   例如，在配置了 PTP 事件的集群中，以下 YAML 片断演示了一个云事件 sidecar 部署：

```
containers:
  - name: cloud-event-sidecar
    image: cloud-event-sidecar
    args:
      - "--metrics-addr=127.0.0.1:9091"
      - "--store-path=/store"
```

```
    - "--transport-host=consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043"
    - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043" 1
    - "--api-port=8089"
```

1 PTP Operator 会自动将 NODE_NAME 解析为正在生成 PTP 事件的主机。例如，compute-1.example.com。

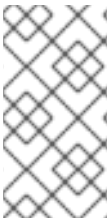
在配置了裸机事件的集群中，在云事件 sidecar 部署 CR 中将 http-event-publishers 字段设置为 hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043。

2. 将 consumer-events-subscription-service 服务与事件消费者应用程序一起部署。例如：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

20.3.5. 安装 AMQ 消息传递总线

要在节点上的发布程序与订阅者之间传递 PTP 快速事件通知，您必须安装和配置 AMQ 消息传递总线，以便在节点上本地运行。要使用 AMQ 消息传递，您必须安装 AMQ Interconnect Operator。:_mod-docs-content-type: SNIPPET



注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

先决条件

- 以具有 cluster-admin 特权的用户身份登录。

流程

- 将 AMQ Interconnect Operator 安装到其自己的 amq-interconnect 命名空间。请参阅添加 Red Hat Integration - AMQ Interconnect Operator。

验证

1. 检查 AMQ Interconnect Operator 是否可用，且所需的 pod 是否正在运行：

```
$ oc get pods -n amq-interconnect
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
amq-interconnect-645db76c76-k8ghs      1/1    Running   0        23h
interconnect-operator-5cb5fc7cc-4v7qm   1/1    Running   0        23h
```

2. 检查所需的 linuxptp-daemon PTP 事件制作者 pod 是否在openshift-ptp 命名空间中运行。

```
$ oc get pods -n openshift-ptp
```

输出示例

```
NAME                 READY   STATUS    RESTARTS      AGE
linuxptp-daemon-2t78p   3/3     Running   0             12h
linuxptp-daemon-k8n88   3/3     Running   0             12h
```

20.3.6. 使用 REST API 将 DU 应用程序订阅到 PTP 事件

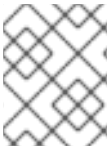
使用资源地址 /cluster/node/<node_name>/ptp 将应用程序订阅到 PTP 事件，其中 <node_name> 是运行 DU 应用程序的集群节点。

在单独的 DU 应用程序 pod 中部署 cloud-event-consumer DU 应用程序容器和cloud-event-proxy sidecar 容器。cloud-event-consumer DU 应用程序订阅应用程序 Pod 中的cloud-event-proxy 容器。

使用以下 API 端点，将 cloud-event-consumer DU 应用程序订阅到 PTP 事件，这些事件由cloud-event-proxy 容器发布，位于 DU 应用程序 pod 中的 http://localhost:8089/api/ocloudNotifications/v1/：

- /api/ocloudNotifications/v1/subscriptions
  - POST ：创建新订阅
  - GET ：删除订阅列表
- /api/ocloudNotifications/v1/subscriptions/<subscription_id>
  - GET ：返回指定订阅 ID 的详情
- /api/ocloudNotifications/v1/health
  - GET：返回 ocloudNotifications API 的健康状况
- api/ocloudNotifications/v1/publishers
  - GET ：为集群节点返回数组 os-clock-sync-state、ptp-clock-class-change、lock-state 和 gnss-sync-status 消息

- /api/ocloudnotifications/v1/<resource_address>/CurrentState
  - GET ：返回以下事件类型的当前状态： os-clock-sync-state、ptp-clock-class-change、lock-state 或 gnss-state-change 事件



注意

9089 是在应用程序 Pod 中部署的 cloud-event-consumer 容器的默认端口。您可以根据需要为 DU 应用程序配置不同的端口。

20.3.6.1. PTP 事件 REST API 参考

使用 PTP 事件通知 REST API 将集群应用程序订阅到父节点上生成的 PTP 事件。

20.3.6.1.1. api/ocloudNotifications/v1/subscriptions

HTTP 方法
GET api/ocloudNotifications/v1/subscriptions

描述
返回订阅列表。如果订阅存在，则返回 200 OK 状态代码以及订阅列表。

API 响应示例

```
[
 {
  "id": "75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
  "resource": "/cluster/node/compute-1.example.com/ptp"
 }
]
```

HTTP 方法
POST api/ocloudNotifications/v1/subscriptions

描述
创建新订阅。如果订阅成功创建，或者已存在，则返回 201 Created 状态代码。

表 20.12. 查询参数

参数	类型
subscription	data

有效负载示例

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

20.3.6.1.2. api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP 方法
GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

描述
返回 ID 为 <subscription_id> 的订阅详情

表 20.13. 查询参数

参数	类型
<subscription_id>	string

API 响应示例

```
{
  "id":"48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation":"http://localhost:8089/api/ocloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "resource":"/cluster/node/compute-1.example.com/ptp"
}
```

20.3.6.1.3. api/ocloudNotifications/v1/health

HTTP 方法
GET api/ocloudNotifications/v1/health/

描述
返回 ocloudNotifications REST API 的健康状况。

API 响应示例

```
OK
```

20.3.6.1.4. api/ocloudNotifications/v1/publishers

HTTP 方法
GET api/ocloudNotifications/v1/publishers

描述
返回集群节点的 os-clock-sync-state、ptp-clock-class-change、lock-state 和 gnss-sync-status 详情的数组。当相关的设备状态改变时，系统会生成通知。

- **os-clock-sync-state 通知描述了主机操作系统时钟同步状态。可以是 LOCKED 或 FREERUN 状态。**
- **ptp-clock-class-change 通知描述了 PTP 时钟类的当前状态。**
- **lock-state 通知描述了 PTP 设备锁定状态的当前状态。可以处于 LOCKED、HOLDOVER 或 FREERUN 状态。**

- gNSS-sync-status 通知描述了与外部 GNSS 时钟信号相关的 GPS 同步状态。可以是LOCKED 或 FREERUN 状态。

您可以组合使用设备同步状态订阅，以提供有关系统总体同步健康状况的详细视图。

API 响应示例

```
[
  {
    "id": "0fa415ae-a3cf-4299-876a-589438bacf75",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/0fa415ae-a3cf-4299-876a-589438bacf75",
    "resource": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state"
  },
  {
    "id": "28cd82df-8436-4f50-bbd9-7a9742828a71",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/28cd82df-8436-4f50-bbd9-7a9742828a71",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change"
  },
  {
    "id": "44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state"
  },
  {
    "id": "778da345d-4567-67b0-a43f0-rty885a456",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/778da345d-4567-67b0-a43f0-rty885a456",
    "resource": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status"
  }
]
```

您可以在 cloud-event-proxy 容器的日志中找到 os-clock-sync-state、ptp-clock-class-change、lock-state 和 gnss-sync-status 事件。例如：

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

os-clock-sync-state 事件示例

```
{
  "id":"c8a784d1-5f4a-4c16-9a81-a3b4313affe5",
  "type":"event.sync.sync-status.os-clock-sync-state-change",
  "source":"/cluster/compute-1.example.com/ptp/CLOCK_REALTIME",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.906277159Z",
  "data":{
    "version":"v1",
    "values":[
```

```
        {
          "resource":"/sync/sync-status/os-clock-sync-state",
          "dataType":"notification",
          "valueType":"enumeration",
          "value":"LOCKED"
        },
        {
          "resource":"/sync/sync-status/os-clock-sync-state",
          "dataType":"metric",
          "valueType":"decimal64.3",
          "value":"-53"
        }
      ]
    }
}
```

ptp-clock-class-change 事件示例

```
{
  "id":"69eddb52-1650-4e56-b325-86d44688d02b",
  "type":"event.sync.ptp-status.ptp-clock-class-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.147100033Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/ptp-clock-class-change",
        "dataType":"metric",
        "valueType":"decimal64.3",
        "value":"135"
      }
    ]
  }
}
```

lock-state 事件示例

```
{
  "id":"305ec18b-1472-47b3-aadd-8f37933249a9",
  "type":"event.sync.ptp-status.ptp-state-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.467684081Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/lock-state",
        "dataType":"notification",
        "valueType":"enumeration",
        "value":"LOCKED"
      },
      {
```

```
          "resource":"/sync/ptp-status/lock-state",
          "dataType":"metric",
          "valueType":"decimal64.3",
          "value":"62"
        }
      ]
    }
}
```

gnss-sync-status 事件示例

```
{
  "id": "435e1f2a-6854-4555-8520-767325c087d7",
  "type": "event.sync.gnss-status.gnss-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "dataContentType": "application/json",
  "time": "2023-09-27T19:35:33.42347206Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "5"
      }
    ]
  }
}
```

20.3.6.1.5. api/ocloudNotifications/v1/<resource_address>/CurrentState

HTTP 方法
GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState

描述
配置 CurrentState API 端点，以返回 os-clock-sync-state、ptp-clock-class-change、lock-state 事件的当前状态。

- **os-clock-sync-state 通知描述了主机操作系统时钟同步状态。可以是 LOCKED 或 FREERUN 状态。**

- **ptp-clock-class-change 通知描述了 PTP 时钟类的当前状态。**
- **lock-state 通知描述了 PTP 设备锁定状态的当前状态。可以处于 LOCKED、HOLDOVER 或 FREERUN 状态。**

表 20.14. 查询参数

参数	类型
<resource_address>	string

lock-state API 响应示例

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

os-clock-sync-state API 响应示例

```
{
  "specversion": "0.3",
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "notification",
```

```
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "27"
      }
    ]
  }
}
```

ptp-clock-class-change API 响应示例

```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```

20.3.7. 监控 PTP 快速事件指标

您可以从运行 linuxptp-daemon 的集群节点监控 PTP 快速事件指标。您还可以使用预先配置和自我更新的 Prometheus 监控堆栈来监控 OpenShift Container Platform Web 控制台中的 PTP 快速事件指标。

先决条件

- 安装 OpenShift Container Platform CLI oc。
- 以具有 cluster-admin 特权的用户身份登录。
- 在具有 PTP 功能硬件的节点上安装和配置 PTP Operator。

流程

1. 运行以下命令，为节点启动 debug pod：

   ```
   $ oc debug node/<node_name>
   ```

2. 检查 linuxptp-daemon 容器公开的 PTP 指标。例如，运行以下命令：

```
sh-4.4# curl http://localhost:9091/metrics
```

输出示例

```
# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",status="success"} 27
```

3. 要在 OpenShift Container Platform web 控制台中查看 PTP 事件，请复制您要查询的 PTP 指标的名称，如 openshift_ptp_offset_ns。
4. 在 OpenShift Container Platform web 控制台中点 Observe → Metrics。
5. 将 PTP 指标名称粘贴到 Expression 字段中，然后点 Run query。

其他资源

- 管理指标

20.3.8. PTP 快速事件指标参考

下表描述了运行 linuxptp-daemon 服务的集群节点可用的 PTP 快速事件指标。

注意

以下一些指标仅适用于 PTP grandmaster 时钟(T-GM)。

表 20.15. PTP 快速事件指标

指标	描述	Example
openshift_ptp_clock_class	返回接口的 PTP 时钟类。对于 PTP 时钟类的可能值为：6 (**LOCKED**), 7 (**PRC UNLOCKED IN-SPEC**), 52 (**PRC UNLOCKED OUT-OF-SPEC**), 187 (**PRC UNLOCKED OUT-OF-SPEC**), 135 (**T-BC HOLDOVER IN-SPEC**), 165 (**T-BC HOLDOVER OUT-OF-SPEC**), 248 (**DEFAULT**), 或 255 (**SLAVE ONLY CLOCK**)。仅适用于 T-GM 时钟。	**{node="compute-1.example.com",process="ptp4l"} 6**
openshift_ptp_clock_state	返回接口的当前 PTP 时钟状态。PTP 时钟状态的可能值为 **FREERUN**、**LOCKED** 或 **HOLDOVER**。	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**

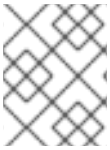
指标	描述	Example
openshift_ptp_delay_ns	返回主时钟发送计时数据包和接收计时数据包之间的延迟（以纳秒为单位）。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_frequency_adjustment_ns	以纳秒为单位返回 2 PTP 时钟之间的频率调整。例如，在上游时钟和 NIC 之间，系统时钟和 NIC 之间，或在 PTP 硬件时钟(**phc**)和 NIC 之间。仅适用于 T-GM 时钟。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_frequency_status	返回 NIC 的数字阶段锁定循环(DPLL)频率的当前状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_phase_status	返回 NIC 的 DPLL 阶段的状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_interface_role	返回为接口配置的 PTP 时钟角色。可能的值包括 0 (**PASSIVE**), 1 (**SLAVE**), 2 (**MASTER**), 3 (**FAULTY**), 4 (**UNKNOWN**), or 5 (**LISTENING**).	**{iface="ens2f0", node="compute-1.example.com", process="ptp4l"} 2**
openshift_ptp_max_offset_ns	返回 2 时钟或接口之间的最大偏移量（以纳秒为单位）。例如，在上游 GNSS 时钟和 NIC (**ts2phc**)之间，或在 PTP 硬件时钟(**phc**)和系统时钟(**phc2sys**)之间。仅适用于 T-GM 时钟。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	返回 DPLL 时钟或 GNSS 时钟源和 NIC 硬件时钟之间的偏移量。仅适用于 T-GM 时钟。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**
openshift_ptp_process_restart_count	返回 **ptp4l** 进程重启的次数。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_process_status	返回显示 PTP 进程是否正在运行的状态代码。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**

指标	描述	Example
openshift_ptp_threshold	为 **HoldOverTimeout**, **MaxOffsetThreshold**, 和 **MinOffsetThreshold** 返回值。 • **holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。 • **maxOffsetThreshold** 和 **minOffsetThreshold** 是以纳秒为单位的偏移值，它比较 **CLOCK_REALTIME** (**phc2sys**) 的值，或您在 **PtpConfig** CR 中为 NIC 配置的 r master offset (**ptp4l**) 的值。	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**
openshift_ptp_pps_status	返回 NIC 1PPS 连接的当前状态。您可以使用 1PPS 连接在连接的 NIC 之间同步计时。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**
openshift_ptp_nmea_status	返回 NMEA 连接的当前状态。NMEA 是 1PPS NIC 连接使用的协议。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_gnss_status	返回全局导航 Satellite 系统(GNSS)连接的当前状态。GNSS 在全局范围内提供基于 satellite 的位置、导航和计时服务。可能的值包括 0 (**NOFIX**), 1 (**DEAD RECKONING ONLY**), 2 (**2D-FIX**), 3 (**3D-FIX**), 4 (**GPS+DEAD RECKONING FIX**), 5, (**TIME ONLY FIX**).	**{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3**
openshift_ptp_ha_profile_status	当不同 NIC 上有多个时间源时，返回高可用性系统时钟的当前状态。可能的值为 0 (**INACTIVE**) 和 1 (**ACTIVE**)。	**{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0**

20.4. 开发 PTP 事件消费者应用程序

在裸机集群节点上开发使用 Precision Time Protocol (PTP) 事件的消费者应用程序时，您需要在单独的应用程序 pod 中部署消费者应用程序和 cloud-event-proxy 容器。cloud-event-proxy 容器从 PTP Operator pod 接收事件，并将其传递给消费者应用程序。消费者应用使用 REST API 订阅 cloud-event-proxy 容器中发布的事件。

有关部署 PTP 事件通知的更多信息，请参阅关于 PTP 快速事件通知框架。



注意

以下信息提供了开发使用 PTP 事件的消费者应用程序的一般指导。完整的事件消费者应用示例超出了此信息的范围。

20.4.1. PTP 事件消费者应用程序参考

PTP 事件消费者应用程序需要以下功能：

1. 使用 POST 处理程序运行的 Web 服务，以接收云原生 PTP 事件 JSON 有效负载
2. 订阅 PTP 事件制作者的 createSubscription 功能
3. getCurrentState 功能轮询 PTP 事件制作者的当前状态

以下示例 Go 片断演示了这些要求：

Go 中的 PTP 事件消费者服务器功能示例

```
func server() {
  http.HandleFunc("/event", getEvent)
  http.ListenAndServe("localhost:8989", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
  defer req.Body.Close()
  bodyBytes, err := io.ReadAll(req.Body)
  if err != nil {
    log.Errorf("error reading event %v", err)
  }
  e := string(bodyBytes)
  if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
  } else {
    w.WriteHeader(http.StatusNoContent)
  }
}
```

Go 中的 PTP 事件 createSubscription 功能示例

```
import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state")
(1)
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-
change")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
  var status int
     apiPath:= "/api/ocloudNotifications/v1/"
     localAPIAddr:=localhost:8989 // vDU service API address
     apiAddr:= "localhost:8089" // event framework API address
```

```
  subURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: apiAddr
    Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
  endpointURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: localAPIAddr,
    Path: "event"}}

  sub = v1pubsub.NewPubSub(endpointURL, resourceAddress)
  var subB []byte

  if subB, err = json.Marshal(&sub); err == nil {
    rc := restclient.New()
    if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
      err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL,
status)
    } else {
      err = json.Unmarshal(subB, &sub)
    }
  } else {
    err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
  }
  return
}
```

1 将 <node_name> 替换为正在生成 PTP 事件的节点 FQDN。例如，compute-1.example.com。

Go 中的 PTP 事件消费者 getCurrentState 功能示例

```
//Get PTP event state for the resource
func getCurrentState(resource string) {
  //Create publisher
  url := &types.URI{URL: url.URL{Scheme: "http",
    Host: localhost:8989,
    Path: fmt.SPrintf("/api/ocloudNotifications/v1/%s/CurrentState",resource}}
  rc := restclient.New()
  status, event := rc.Get(url)
  if status != http.StatusOK {
    log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
  } else {
    log.Debugf("Got CurrentState: %s ", event)
  }
}
```

20.4.2. 引用 cloud-event-proxy 部署和服务 CR

在部署 PTP 事件消费者应用程序时，使用 cloud-event-proxy 部署和订阅者服务 CR 示例作为参考。

注意

HTTP 传输是 PTP 和裸机事件的默认传输。在可能的情况下，使用 HTTP 传输而不是 AMQP 用于 PTP 和裸机事件。AMQ Interconnect 于 2024 年 6 月 30 日结束生命周期（EOL）。AMQ Interconnect 的延长生命周期支持 (ELS) 于 2029 年 11 月 29 日结束。如需更多信息，请参阅 Red Hat AMQ Interconnect 支持状态。

使用 HTTP 传输引用 cloud-event-proxy 部署

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-consumer-deployment
  namespace: <namespace>
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      serviceAccountName: sidecar-consumer-sa
      containers:
        - name: event-subscriber
          image: event-subscriber-app
        - name: cloud-event-proxy-as-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
            - "--api-port=8089"
          env:
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: NODE_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.hostIP
              volumeMounts:
                - name: pubsubstore
                  mountPath: /store
          ports:
            - name: metrics-port
              containerPort: 9091
            - name: sub-port
              containerPort: 9043
          volumes:
            - name: pubsubstore
              emptyDir: {}
```

使用 AMQ 传输引用 cloud-event-proxy 部署

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-event-proxy-sidecar
  namespace: cloud-events
  labels:
    app: cloud-event-proxy
spec:
  selector:
    matchLabels:
      app: cloud-event-proxy
  template:
    metadata:
      labels:
        app: cloud-event-proxy
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
        - name: cloud-event-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=amqp://router.router.svc.cluster.local"
            - "--api-port=8089"
          env:
            - name: <node_name>
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: <node_ip>
              valueFrom:
                fieldRef:
                  fieldPath: status.hostIP
          volumeMounts:
            - name: pubsubstore
              mountPath: /store
          ports:
            - name: metrics-port
              containerPort: 9091
            - name: sub-port
              containerPort: 9043
          volumes:
            - name: pubsubstore
              emptyDir: {}
```

参考 cloud-event-proxy 订阅者服务

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
```

```
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

20.4.3. cloud-event-proxy sidecar REST API 中的 PTP 事件

PTP 事件消费者应用程序可以为以下 PTP 时间事件轮询 PTP 事件制作者。

表 20.16. cloud-event-proxy sidecar 中可用的 PTP 事件

资源 URI	描述
/cluster/node/<node_name>/sync/ptp-status/lock-state	描述 PTP 设备锁定状态的当前状态。可以处于 LOCKED、HOLDOVER 或 FREERUN 状态。
/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state	描述主机操作系统时钟同步状态。可以是 LOCKED 或 FREERUN 状态。
/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change	描述 PTP 时钟类的当前状态。

20.4.4. 将消费者应用程序订阅到 PTP 事件

在 PTP 事件消费者应用程序可以轮询事件前，您需要将应用程序订阅到事件制作者。

20.4.4.1. 订阅 PTP 锁定状态事件

要为 PTP lock-state 事件创建一个订阅，使用以下 payload 向云事件 API（位于 http://localhost:8081/api/ocloudNotifications/v1/subscriptions）发送一个 POST 操作：

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

响应示例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-
```

```
4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

20.4.4.2. 订阅 PTP os-clock-sync-state 事件

要为 PTP os-clock-sync-state 事件创建一个订阅，使用以下 payload 向云事件 API（位于 http://localhost:8081/api/ocloudNotifications/v1/subscriptions）发送一个 POST 操作：

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

响应示例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-
4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

20.4.4.3. 订阅 PTP ptp-clock-class-change 事件

要为 PTP ptp-clock-class-change 事件创建一个订阅，使用以下 payload 向云事件 API（位于 http://localhost:8081/api/ocloudNotifications/v1/subscriptions）发送一个 POST 操作：

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

响应示例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-
4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

20.4.5. 获取当前的 PTP 时钟状态

要获取节点的当前 PTP 状态，请发送 GET 操作到以下事件 REST API 之一：

- http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState

- http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState
- http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState

响应是一个云原生事件 JSON 对象。例如：

lock-state API 响应示例

```
{
 "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
 "type": "event.sync.ptp-status.ptp-state-change",
 "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
 "dataContentType": "application/json",
 "time": "2023-01-10T02:41:57.094981478Z",
 "data": {
  "version": "v1",
  "values": [
   {
    "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
    "dataType": "notification",
    "valueType": "enumeration",
    "value": "LOCKED"
   },
   {
    "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
    "dataType": "metric",
    "valueType": "decimal64.3",
    "value": "29"
   }
  ]
 }
}
```

20.4.6. 验证 PTP 事件消费者应用程序是否收到事件

验证应用程序 pod 中的 cloud-event-proxy 容器是否接收 PTP 事件。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 已安装并配置了 PTP Operator。

流程

1. 获取活跃的 linuxptp-daemon pod 列表。运行以下命令:

```
$ oc get pods -n openshift-ptp
```

输出示例

```
NAME                READY   STATUS    RESTARTS   AGE
linuxptp-daemon-2t78p   3/3     Running   0          8h
linuxptp-daemon-k8n88   3/3     Running   0          8h
```

2. 运行以下命令，访问所需的消费者端的 cloud-event-proxy 容器的指标：

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl
127.0.0.1:9091/metrics
```

其中：

<linuxptp-daemon>

指定您要查询的 pod，例如 linuxptp-daemon-2t78p。

输出示例

```
# HELP cne_transport_connections_resets Metric to get number of connection
resets
# TYPE cne_transport_connections_resets gauge
cne_transport_connection_reset 1
# HELP cne_transport_receiver Metric to get number of receiver created
# TYPE cne_transport_receiver gauge
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 2
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 2
# HELP cne_transport_sender Metric to get number of sender created
# TYPE cne_transport_sender gauge
cne_transport_sender{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 1
cne_transport_sender{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 1
# HELP cne_events_ack Metric to get number of events produced
# TYPE cne_events_ack gauge
cne_events_ack{status="success",type="/cluster/node/compute-
1.example.com/ptp"} 18
cne_events_ack{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP cne_events_transport_published Metric to get number of events published
by the transport
# TYPE cne_events_transport_published gauge
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_transport_received Metric to get number of events received  by
the transport
# TYPE cne_events_transport_received gauge
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_received{address="/cluster/node/compute-
```

```
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_api_published Metric to get number of events published by the
rest api
# TYPE cne_events_api_published gauge
cne_events_api_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 19
cne_events_api_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 19
# HELP cne_events_received Metric to get number of events received
# TYPE cne_events_received gauge
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/ptp"} 18
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes
being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP
status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

第 21 章 外部 DNS OPERATOR

21.1. 外部 DNS OPERATOR 发行注记

External DNS Operator 部署并管理 ExternalDNS，以便为从外部 DNS 供应商到 OpenShift Container Platform 的服务和路由提供名称解析。

本发行注记介绍了 OpenShift Container Platform 中外部 DNS Operator 的开发。

21.1.1. 外部 DNS Operator 1.2.0

以下公告可用于外部 DNS Operator 版本 1.2.0 ：

- RHEA-2022:5867 ExternalDNS Operator 1.2 operator/operand containers

21.1.1.1. 新功能

- External DNS Operator 现在支持 AWS 共享 VPC。如需更多信息，请参阅使用共享 VPC 在不同的 AWS 帐户中创建 DNS 记录。

21.1.1.2. 程序错误修复

- 操作对象的更新策略从 Rolling 改为 Recreate。（OCPBUGS-3630）

21.1.2. 外部 DNS Operator 1.1.1

以下公告可用于外部 DNS Operator 版本 1.1.1：

- RHEA-2024:0536 ExternalDNS Operator 1.1 operator/operand containers

21.1.3. 外部 DNS Operator 1.1.0

此发行版本包含来自上游项目 0.13.1 版本的操作对象的变基。以下公告可用于外部 DNS Operator 版本 1.1.0：

- RHEA-2022:9086-01 ExternalDNS Operator 1.1 operator/operand containers

21.1.3.1. 程序错误修复

- 在以前的版本中，ExternalDNS Operator 为卷强制有一个空的 defaultMode 值，这会导致因为与 OpenShift API 冲突而造成恒定的更新。现在，defaultMode 值不会被强制，操作对象部署不会持续更新。（OCPBUGS-2793）

21.1.4. 外部 DNS Operator 1.0.1

以下公告可用于外部 DNS Operator 版本 1.0.1：

- RHEA-2024:0537 ExternalDNS Operator 1.0 operator/operand containers

21.1.5. 外部 DNS Operator 1.0.0

以下公告可用于 External DNS Operator 版本 1.0.0：

- RHEA-2022:5867 ExternalDNS Operator 1.0 operator/operand containers

21.1.5.1. 程序错误修复

- 在以前的版本中，External DNS Operator 在 ExternalDNS 操作对象 pod 部署中发出有关违反 restricted SCC 策略的警告。这个问题已解决。(BZ#2086408)

21.2. OPENSHIFT CONTAINER PLATFORM 中的外部 DNS OPERATOR

External DNS Operator 部署并管理 ExternalDNS，以便为从外部 DNS 供应商到 OpenShift Container Platform 的服务和路由提供名称解析。

21.2.1. 外部 DNS Operator

External DNS Operator 从 olm.openshift.io API 组实现外部 DNS API。External DNS Operator 更新服务、路由和外部 DNS 供应商。

先决条件

- 已安装 yq CLI 工具。

流程

您可以根据 OperatorHub 的要求部署外部 DNS Operator。部署外部 DNS Operator 会创建一个 Subscription 对象。

1. 运行以下命令，检查安装计划的名称：

```
$ oc -n external-dns-operator get sub external-dns-operator -o yaml | yq
'.status.installplan.name'
```

输出示例

```
install-zcvlr
```

2. 运行以下命令，检查安装计划的状态是否为 Complete ：

```
$ oc -n external-dns-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

输出示例

```
Complete
```

3. 运行以下命令，查看 external-dns-operator 部署的状态：

```
$ oc get -n external-dns-operator deployment/external-dns-operator
```

输出示例

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
external-dns-operator   1/1     1            1           23h
```

21.2.2. 查看外部 DNS Operator 日志

您可以使用 oc logs 命令查看外部 DNS Operator 日志。

流程

1. 运行以下命令，查看外部 DNS Operator 的日志：

```
$ oc logs -n external-dns-operator deployment/external-dns-operator -c external-dns-operator
```

21.2.2.1. 外部 DNS Operator 域名限制

External DNS Operator 使用 TXT registry，它为 TXT 记录添加前缀。这可减少 TXT 记录的域名的最大长度。没有对应的 TXT 记录时无法出现 DNS 记录，因此 DNS 记录的域名必须遵循与 TXT 记录相同的限制。例如，一个 <domain_name_from_source> DNS 记录会导致一个 external-dns-<record_type>-<domain_name_from_source> TXT 记录。

外部 DNS Operator 生成的 DNS 记录的域名有以下限制：

记录类型	字符数
CNAME	44
AzureDNS 上的通配符 CNAME 记录	42
A	48
AzureDNS 上的通配符 A 记录	46

如果生成的域名超过任何域名限制，则外部 DNS Operator 日志中会出现以下错误：

```
time="2022-09-02T08:53:57Z" level=error msg="Failure in zone test.example.io. [Id: /hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=error msg="InvalidChangeBatch: [FATAL problem: DomainLabelTooLong (Domain label is too long) encountered with 'external-dns-a-hello-openshift-aaaaaaaaaa-bbbbbbbbbb-ccccccc']\n\tstatus code: 400, request id: e54dfd5a-06c6-47b0-bcb9-a4f7c3a4e0c6"
```

21.3. 在云供应商上安装外部 DNS OPERATOR

您可以在云供应商环境中安装外部 DNS Operator，如 AWS、Azure 和 GCP。

21.3.1. 安装 External DNS Operator

您可以使用 OpenShift Container Platform OperatorHub 安装外部 DNS Operator。

流程

1. 在 OpenShift Container Platform Web 控制台中点 Operators → OperatorHub。
2. 点 External DNS Operator。您可以使用 Filter by keyword 文本框或过滤器列表从 Operator 列表中搜索 External DNS Operator。
3. 选择 **external-dns-operator** 命名空间。
4. 在 External DNS Operator 页面中，点 Install。
5. 在 Install Operator 页面中，确保选择了以下选项：
   a. 将频道更新为 stable-v1。
   b. 安装模式为 A specific name on the cluster。
   c. 安装的命名空间为 **external-dns-operator**。如果命名空间 **external-dns-operator** 不存在，它会在 Operator 安装过程中创建。
   d. 将 Approval Strategy 选为 Automatic 或 Manual。默认情况下，批准策略设置为 Automatic。
   e. 点 Install。

如果选择了 Automatic 更新，Operator Lifecycle Manager(OLM)将自动升级 Operator 的运行实例，而无需任何干预。

如果选择 手动 更新，则 OLM 会创建一个更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。

验证

验证 External DNS Operator 是否在 Installed Operators 仪表板上显示 Status 为 Succeeded。

21.4. 外部 DNS OPERATOR 配置参数

External DNS Operator 包括以下配置参数。

21.4.1. 外部 DNS Operator 配置参数

External DNS Operator 包括以下配置参数：

参数	描述

参数	描述
spec	启用云供应商的类型。 spec: provider: type: AWS (1) aws: credentials: name: aws-access-key (2) (1) 定义可用选项，如 AWS、GCP、Azure 和 Infoblox。 (2) 为云供应商定义 secret 名称。
zones	允许您根据域指定 DNS 区域。如果没有指定区，**External** DNS 资源会发现云供应商帐户中存在的所有区域。 zones: - "myzoneid" (1) (1) 指定 DNS 区域的名称。
domains	允许您根据域指定 AWS 区域。如果没有指定域，**External** DNS 资源会发现云供应商帐户中存在的所有区域。 domains: - filterType: Include (1) matchType: Exact (2) name: "myzonedomain1.com" (3) - filterType: Include matchType: Pattern (4) pattern: ".*\\.otherzonedomain\\.com" (5) (1) 确保 **ExternalDNS** 资源包含域名。 (2) 指示 **ExtrnalDNS** 指示域匹配必须完全匹配，而不是正则表达式匹配。 (3) 定义域的名称。 (4) 在 **ExternalDNS** 资源中设置 **regex-domain-filter** 标志。您可以使用 Regex 过滤器来限制可能的域。 (5) 定义 **ExternalDNS** 资源用来过滤目标区的域的 regex 模式。

参数	描述
source	允许您指定 DNS 记录、**Service** 或 **Route** 的源。

```
source: 1
  type: Service 2
  service:
    serviceType: 3
      - LoadBalancer
      - ClusterIP
  labelFilter: 4
    matchLabels:
      external-dns.mydomain.org/publish: "yes"
  hostnameAnnotation: "Allow" 5
  fqdnTemplate:
  - "{{.Name}}.myzonedomain.com" 6
```

1. 定义 DNS 记录源的设置。
2. **ExternalDNS** 资源使用 **Service** 类型作为创建 DNS 记录的源。
3. 在 **ExternalDNS** 资源中设置 **service-type-filter** 标志。**serviceType** 包含以下字段：
   - **默认:LoadBalancer**
   - **预期:ClusterIP**
   - **NodePort**
   - **LoadBalancer**
   - **ExternalName**
4. 确保控制器只考虑与标签过滤器匹配的资源。
5. **hostnameAnnotation** 的默认值为 **Ignore**，它指示 **ExternalDNS** 使用字段 **fqdnTemplates** 中指定的模板生成 DNS 记录。当值是 **Allow**，DNS 记录根据 **external-dns.alpha.kubernetes.io/hostname** 注解中指定的值生成。
6. 外部 DNS Operator 使用一个字符串从没有定义主机名的源生成 DNS 名称，或者与 fake 源配对时添加主机名后缀。

```
source:
  type: OpenShiftRoute 1
  openshiftRouteOptions:
    routerName: default 2
    labelFilter:
      matchLabels:
        external-dns.mydomain.org/publish: "yes"
```

1. 创建 DNS 记录。
2. 如果源类型是 **OpenShiftRoute**，您可以传递 Ingress Controller 名称。**ExternalDNS** 资源使用 Ingress Controller 的规范名称作为 CNAME 记录的目标。

参数	描述

21.5. 在 AWS 上创建 DNS 记录

您可以使用外部 DNS Operator 在 AWS 和 AWS GovCloud 上创建 DNS 记录。

21.5.1. 使用 Red Hat External DNS Operator 在 AWS 公共托管区中创建 DNS 记录

您可以使用 Red Hat External DNS Operator 在 AWS 公共托管区上创建 DNS 记录。您可以使用相同的说明在 AWS GovCloud 的托管区上创建 DNS 记录。

流程

1. 检查用户。用户必须有权访问 **kube-system** 命名空间。如果没有凭证，您可以从 **kube-system** 命名空间中获取凭证，以使用云供应商客户端：

```
$ oc whoami
```

输出示例

```
system:admin
```

2. 从 **kube-system** 命名空间中存在的 aws-creds secret 中获取值。

```
$ export AWS_ACCESS_KEY_ID=$(oc get secrets aws-creds -n kube-system  --
template={{.data.aws_access_key_id}} | base64 -d)
$ export AWS_SECRET_ACCESS_KEY=$(oc get secrets aws-creds -n kube-system  --
template={{.data.aws_secret_access_key}} | base64 -d)
```

3. 获取路由来检查域：

```
$ oc get routes --all-namespaces | grep console
```

输出示例

```
openshift-console          console             console-openshift-
console.apps.testextdnsoperator.apacshift.support                  console
https   reencrypt/Redirect   None
openshift-console          downloads           downloads-openshift-
console.apps.testextdnsoperator.apacshift.support                  downloads
http    edge/Redirect        None
```

4. 获取 dns zones 列表以查找与之前找到的路由域对应的 dns 区域：

```
$ aws route53 list-hosted-zones | grep testextdnsoperator.apacshift.support
```

输出示例

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

5. 为路由源创建 ExternalDNS 资源：

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws 1
spec:
  domains:
  - filterType: Include 2
    matchType: Exact 3
    name: testextdnsoperator.apacshift.support 4
  provider:
    type: AWS 5
  source: 6
    type: OpenShiftRoute 7
    openshiftRouteOptions:
      routerName: default 8
EOF
```

1. 定义外部 DNS 资源的名称。
2. 默认情况下，所有托管区都被选为潜在的目标。您可以包括需要的托管区。
3. 目标区的域匹配必须是完全准确的（与正则表达式匹配不同）。
4. 指定您要更新的区域的确切域。路由的主机名必须是指定域的子域。
5. 定义 AWS Route53 DNS 供应商。
6. 定义 DNS 记录源的选项。
7. 定义 OpenShift 路由资源，作为在之前指定的 DNS 供应商中创建的 DNS 记录来源。
8. 如果源是 OpenShiftRoute，您可以传递 OpenShift Ingress Controller 名称。外部 DNS Operator 在创建 CNAME 记录时，选择该路由器的规范主机名作为目标。

6. 使用以下命令，检查为 OCP 路由创建的记录：

```
$ aws route53 list-resource-record-sets --hosted-zone-id Z02355203TNN1XXXX1J6O --
query "ResourceRecordSets[?Type == 'CNAME']" | grep console
```

21.5.2. 使用共享 VPC 在不同的 AWS 帐户中创建 DNS 记录

您可以使用 ExternalDNS Operator 使用共享 Virtual Private Cloud (VPC)在不同的 AWS 帐户中创建 DNS 记录。通过使用共享 VPC，组织可将资源从多个项目连接到通用 VPC 网络。然后，机构可以使用 VPC 共享在多个 AWS 帐户间使用单个 Route 53 实例。

先决条件

- 您已创建了两个 Amazon AWS 帐户：一个 VPC 和配置了 Route 53 私有托管区（帐户 A），另一个用于安装集群（帐户 B）。

- 您已创建了具有帐户 B 的相应权限的 IAM 策略和 IAM 角色，以便在帐户 A 的 Route 53 托管区中创建 DNS 记录。
- 您已在帐户 B 上安装了帐户 A 的集群。
- 您已在帐户 B 的集群中安装了 ExternalDNS Operator。

流程

1. 运行以下命令，获取您创建的 IAM 角色的 Role ARN，以允许帐户 B 访问帐户 A 的 Route 53 托管区：

```
$ aws --profile account-a iam get-role --role-name user-rol1 | head -1
```

输出示例

```
ROLE arn:aws:iam::1234567890123:role/user-rol1 2023-09-14T17:21:54+00:00 3600 /
AROA3SGB2ZRKRT5NISNJN user-rol1
```

2. 运行以下命令，找到用于帐户 A 凭证的私有托管区：

```
$ aws --profile account-a route53 list-hosted-zones | grep
testextdnsoperator.apacshift.support
```

输出示例

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

3. 运行以下命令来创建 ExternalDNS 对象：

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws
spec:
  domains:
  - filterType: Include
    matchType: Exact
    name: testextdnsoperator.apacshift.support
  provider:
    type: AWS
    aws:
      assumeRole:
        arn: arn:aws:iam::12345678901234:role/user-rol1 (1)
  source:
    type: OpenShiftRoute
    openshiftRouteOptions:
      routerName: default
EOF
```

(1) 指定 Role ARN，以便在帐户 A 中创建 DNS 记录。

4. 使用以下命令，检查为 OpenShift Container Platform (OCP) 路由创建的记录：

```
$ aws --profile account-a route53 list-resource-record-sets --hosted-zone-id
Z02355203TNN1XXXX1J6O --query "ResourceRecordSets[?Type == 'CNAME']" | grep
console-openshift-console
```

21.6. 在 AZURE 上创建 DNS 记录

您可以使用外部 DNS Operator 在 Azure 上创建 DNS 记录。

21.6.1. 在 Azure 公共 DNS 区域中创建 DNS 记录

您可以使用 External DNS Operator 在 Azure 公共 DNS 区域上创建 DNS 记录。

先决条件

- 您必须具有管理员特权。
- admin 用户必须有权访问 kube-system 命名空间。

流程

1. 运行以下命令，从 kube-system 命名空间中获取凭证以使用云供应商客户端：

```
$ CLIENT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_client_id}} | base64 -d)
$ CLIENT_SECRET=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_client_secret}} | base64 -d)
$ RESOURCE_GROUP=$(oc get secrets azure-credentials  -n kube-system  --
template={{.data.azure_resourcegroup}} | base64 -d)
$ SUBSCRIPTION_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_subscription_id}} | base64 -d)
$ TENANT_ID=$(oc get secrets azure-credentials  -n kube-system  --template=
{{.data.azure_tenant_id}} | base64 -d)
```

2. 运行以下命令来登录到 Azure：

```
$ az login --service-principal -u "${CLIENT_ID}" -p "${CLIENT_SECRET}" --tenant
"${TENANT_ID}"
```

3. 运行以下命令来获取路由列表：

```
$ oc get routes --all-namespaces | grep console
```

输出示例

```
openshift-console          console            console-openshift-
console.apps.test.azure.example.com                       console            https
reencrypt/Redirect     None
openshift-console          downloads          downloads-openshift-
console.apps.test.azure.example.com                       downloads          http
edge/Redirect          None
```

4. 运行以下命令，获取 DNS 区域列表：

```
$ az network dns zone list --resource-group "${RESOURCE_GROUP}"
```

5. 创建一个 YAML 文件，如 external-dns-sample-azure.yaml，该文件定义 ExternalDNS 对象：

external-dns-sample-azure.yaml 文件示例

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-azure 1
spec:
  zones:
  - "/subscriptions/1234567890/resourceGroups/test-azure-xxxxx-
rg/providers/Microsoft.Network/dnszones/test.azure.example.com" 2
  provider:
    type: Azure 3
  source:
    openshiftRouteOptions: 4
      routerName: default 5
    type: OpenShiftRoute 6
```

1. 指定外部 DNS 名称。
2. 定义区域 ID。
3. 定义提供程序类型。
4. 您可以定义 DNS 记录源的选项。
5. 如果源类型是 OpenShiftRoute，您可以传递 OpenShift Ingress Controller 名称。外部 DNS 在创建 CNAME 记录时，选择该路由器的规范主机名作为目标。
6. 将 route 资源定义为 Azure DNS 记录的来源。

6. 运行以下命令，检查为 OpenShift Container Platform 路由创建的 DNS 记录：

```
$ az network dns record-set list -g "${RESOURCE_GROUP}"  -z
test.azure.example.com | grep console
```

注意

要在私有 Azure DNS 上的私有托管区上创建记录，您需要在 zones 字段中指定私有区，用于在 ExternalDNS 容器参数中将供应商类型填充到 azure-private-dns。

21.7. 在 GCP 上创建 DNS 记录

您可以使用外部 DNS Operator 在 GCP 上创建 DNS 记录。

21.7.1. 在 GCP 公共管理区上创建 DNS 记录

您可以使用 External DNS Operator 在 GCP 公共受管区上创建 DNS 记录。

先决条件

- 您必须具有管理员特权。

流程

1. 运行以下命令，将 gcp-credentials secret 复制到encoded-gcloud.json 文件中：

```
$ oc get secret gcp-credentials -n kube-system --template='{{$v := index .data
"service_account.json"}}{{$v}}' | base64 -d - > decoded-gcloud.json
```

2. 运行以下命令导出 Google 凭证：

```
$ export GOOGLE_CREDENTIALS=decoded-gcloud.json
```

3. 使用以下命令激活您的帐户：

```
$ gcloud auth activate-service-account  <client_email as per decoded-gcloud.json> --
key-file=decoded-gcloud.json
```

4. 运行以下命令来设置项目：

```
$ gcloud config set project <project_id as per decoded-gcloud.json>
```

5. 运行以下命令来获取路由列表：

```
$ oc get routes --all-namespaces | grep console
```

输出示例

```
openshift-console       console            console-openshift-
console.apps.test.gcp.example.com                  console        https
reencrypt/Redirect     None
openshift-console       downloads          downloads-openshift-
console.apps.test.gcp.example.com                  downloads      http    edge/Redirect
None
```

6. 运行以下命令来获取受管区列表：

```
$ gcloud dns managed-zones list | grep test.gcp.example.com
```

输出示例

```
qe-cvs4g-private-zone test.gcp.example.com
```

7. 创建一个 YAML 文件，如 external-dns-sample-gcp.yaml，该文件定义 ExternalDNS 对象：

external-dns-sample-gcp.yaml 文件示例

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
 name: sample-gcp 1
spec:
 domains:
  - filterType: Include 2
    matchType: Exact 3
    name: test.gcp.example.com 4
 provider:
  type: GCP 5
 source:
  openshiftRouteOptions: 6
   routerName: default 7
  type: OpenShiftRoute 8
```

1. 指定外部 DNS 名称。
2. 默认情况下，所有托管区都被选为潜在的目标。您可以包含托管区。
3. 目标的域必须与 name 键定义的字符串匹配。
4. 指定您要更新的区域的确切域。路由的主机名必须是指定域的子域。
5. 定义提供程序类型。
6. 您可以定义 DNS 记录源的选项。
7. 如果源类型是 OpenShiftRoute，您可以传递 OpenShift Ingress Controller 名称。外部 DNS 在创建 CNAME 记录时，选择该路由器的规范主机名作为目标。
8. 将 route 资源定义为 GCP DNS 记录的源。

8. 运行以下命令，检查为 OpenShift Container Platform 路由创建的 DNS 记录：

```
$ gcloud dns record-sets list --zone=qe-cvs4g-private-zone | grep console
```

21.8. 在 INFOBLOX 上创建 DNS 记录

您可以使用 External DNS Operator 在 Infoblox 上创建 DNS 记录。

21.8.1. 在 Infoblox 上的公共 DNS 区域中创建 DNS 记录

您可以使用 External DNS Operator 在 Infoblox 上的公共 DNS 区域上创建 DNS 记录。

先决条件

- 您可以访问 OpenShift CLI(oc)。
- 您可以访问 Infoblox UI。

流程

1. 运行以下命令，使用 Infoblox 凭证创建 secret 对象：

```
$ oc -n external-dns-operator create secret generic infoblox-credentials --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_USERNAME=<infoblox_username> --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_PASSWORD=<infoblox_password>
```

2. 运行以下命令来获取路由列表：

```
$ oc get routes --all-namespaces | grep console
```

输出示例

```
openshift-console          console             console-openshift-console.apps.test.example.com                 console             https   reencrypt/Redirect     None
openshift-console          downloads           downloads-openshift-console.apps.test.example.com               downloads           http    edge/Redirect          None
```

3. 创建一个 YAML 文件，如 external-dns-sample-infoblox.yaml，该文件定义 ExternalDNS 对象：

external-dns-sample-infoblox.yaml 文件示例

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-infoblox (1)
spec:
  provider:
    type: Infoblox (2)
    infoblox:
      credentials:
        name: infoblox-credentials
      gridHost: ${INFOBLOX_GRID_PUBLIC_IP}
      wapiPort: 443
      wapiVersion: "2.3.1"
  domains:
  - filterType: Include
    matchType: Exact
    name: test.example.com
  source:
    type: OpenShiftRoute (3)
    openshiftRouteOptions:
      routerName: default (4)
```

(1) 指定外部 DNS 名称。

(2) 定义提供程序类型。

(3) 您可以定义 DNS 记录源的选项。

(4) 如果源类型是 OpenShiftRoute，您可以传递 OpenShift Ingress Controller 名称。外部 DNS 在创建 CNAME 记录时，选择该路由器的规范主机名作为目标。

4. 运行以下命令，在 Infoblox 上创建 ExternalDNS 资源：

```
$ oc create -f external-dns-sample-infoblox.yaml
```

5. 通过 Infoblox UI，检查为 console 路由创建的 DNS 记录：

   a. 点 Data Management → DNS → Zones。

   b. 选择区域名称。

21.9. 在外部 DNS OPERATOR 上配置集群范围代理

配置集群范围代理后，Operator Lifecycle Manager (OLM) 会触发对使用 HTTP_PROXY、HTTPS_PROXY 和 NO_PROXY 环境变量的新内容的所有部署的 Operator 的自动更新。

21.9.1. 信任集群范围代理的证书颁发机构

您可以将外部 DNS Operator 配置为信任集群范围代理的证书颁发机构。

流程

1. 运行以下命令，创建配置映射以在 external-dns-operator 命名空间中包含 CA 捆绑包：

```
$ oc -n external-dns-operator create configmap trusted-ca
```

2. 要将可信 CA 捆绑包注入配置映射中，请运行以下命令将 config.openshift.io/inject-trusted-cabundle=true 标签添加到配置映射中：

```
$ oc -n external-dns-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

3. 运行以下命令更新外部 DNS Operator 的订阅：

```
$ oc -n external-dns-operator patch subscription external-dns-operator --type='json' -p='[{"op": "add", "path": "/spec/config", "value":{"env": [{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}}]'
```

验证

- 部署外部 DNS Operator 后，运行以下命令来验证可信 CA 环境变量是否已添加到 external-dns-operator 部署中：

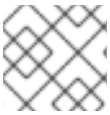
```
$ oc -n external-dns-operator exec deploy/external-dns-operator -c external-dns-operator -- printenv TRUSTED_CA_CONFIGMAP_NAME
```

输出示例

```
trusted-ca
```

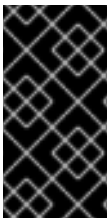
第 22 章 CIDR 范围定义

您必须为以下 CIDR 范围指定非重叠范围。



注意

创建集群后无法更改机器 CIDR 范围。



重要

OVN-Kubernetes 是 OpenShift Container Platform 4.14 及之后的版本中的默认网络供应商，在内部使用以下 IP 地址范围：100.64.0.0/16, 169.254.169.0/29, 100.88.0.0/16, fd98::/64, fd69::/125, 和 fd97::/64。如果您的集群使用 OVN-Kubernetes，请不要在集群或基础架构中的任何其他 CIDR 定义中包含这些 IP 地址范围。

22.1. MACHINE CIDR

在 Machine classless inter-domain routing (CIDR) 字段中，您必须为机器或集群节点指定 IP 地址范围。

默认值为 10.0.0.0/16。这个范围不得与任何连接的网络冲突。

22.2. SERVICE CIDR

在 Service CIDR 字段中，您必须为服务指定 IP 地址范围。范围必须足够大，以适应您的工作负载。该地址块不得与从集群内部访问的任何外部服务重叠。默认为 172.30.0.0/16。

22.3. POD CIDR

在 pod CIDR 字段中，您必须为 pod 指定 IP 地址范围。

pod CIDR 与 clusterNetwork CIDR 和集群 CIDR 相同。范围必须足够大，以适应您的工作负载。该地址块不得与从集群内部访问的任何外部服务重叠。默认为 10.128.0.0/14。您可以在集群安装后扩展范围。

其他资源

- Cluster Network Operator 配置
- 配置集群网络范围

22.4. 主机前缀

在 Host Prefix 字段中，您必须指定分配给调度到各个机器的 pod 的子网前缀长度。主机前缀决定了每台机器的 pod IP 地址池。

例如，如果主机前缀设置为 /23，则每台机器从 pod CIDR 地址范围中分配一个 /23 子网。默认值为 /23，允许 510 集群节点，以及每个节点的 510 个 pod IP 地址。

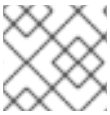
第 23 章 AWS LOAD BALANCER OPERATOR

23.1. AWS LOAD BALANCER OPERATOR 发行注记

AWS Load Balancer (ALB) Operator 部署和管理 **AWSLoadBalancerController** 资源的实例。

本发行注记介绍了 OpenShift Container Platform 中的 AWS Load Balancer Operator 的开发。

如需 AWS Load Balancer Operator 的概述，请参阅 OpenShift Container Platform 中的 AWS Load Balancer Operator。



注意

AWS Load Balancer Operator 目前不支持 AWS GovCloud。

23.1.1. AWS Load Balancer Operator 1.1.1

以下公告可用于 AWS Load Balancer Operator 版本 1.1.1：

- RHEA-2024:0555 Release of AWS Load Balancer Operator 1.1.z on OperatorHub

23.1.2. AWS Load Balancer Operator 1.1.0

AWS Load Balancer Operator 版本 1.1.0 支持 AWS Load Balancer Controller 版本 2.4.4。

以下公告可用于 AWS Load Balancer Operator 版本 1.1.0：

- RHEA-2023:6218 Release of AWS Load Balancer Operator on OperatorHub 增强公告更新

23.1.2.1. 主要变化

- 此发行版本使用 Kubernetes API 版本 0.27.2。

23.1.2.2. 新功能

- AWS Load Balancer Operator 现在支持使用 Cloud Credential Operator 的标准化安全令牌服务 (STS) 流。

23.1.2.3. 程序错误修复

- FIPS 兼容集群必须使用 TLS 版本 1.2。在以前的版本中，AWS Load Balancer Controller 的 Webhook 只接受 TLS 1.3 作为最小版本，从而导致在与 FIPS 兼容的集群中出现以下错误：

  ```
  remote error: tls: protocol version not supported
  ```

  现在，AWS Load Balancer Controller 接受 TLS 1.2 作为最小 TLS 版本，从而解决了这个问题。(OCPBUGS-14846)

23.1.3. AWS Load Balancer Operator 1.0.1

以下公告可用于 AWS Load Balancer Operator 版本 1.0.1：

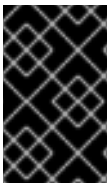
- Release of AWS Load Balancer Operator 1.0.1 on OperatorHub

23.1.4. AWS Load Balancer Operator 1.0.0

现在，AWS Load Balancer Operator 已正式发布。AWS Load Balancer Operator 版本 1.0.0 支持 AWS Load Balancer Controller 版本 2.4.4。

以下公告可用于 AWS Load Balancer Operator 版本 1.0.0：

- RHEA-2023:1954 Release of AWS Load Balancer Operator on OperatorHub 增强公告更新



重要

AWS Load Balancer (ALB) Operator 版本 1.x.x 无法从技术预览版本 0.x.x 自动升级。要从早期版本升级，您必须卸载 ALB 操作对象并删除 **aws-load-balancer-operator** 命名空间。

23.1.4.1. 主要变化

- 此发行版本使用新的 **v1** API 版本。

23.1.4.2. 程序错误修复

- 在以前的版本中，AWS Load Balancer Operator 置备的控制器无法正确将配置用于集群范围代理。这些设置现在对控制器正确应用。(OCPBUGS-4052, OCPBUGS-5295)

23.1.5. 早期版本

AWS Load Balancer Operator 的两个最早版本作为技术预览提供。这些版本不应在生产环境中使用。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

以下公告可用于 AWS Load Balancer Operator 版本 0.2.0 ：

- OperatorHub 增强更新上的 AWS Load Balancer Operator 的 RHEA-2022:9084 发行版本

以下公告可用于 AWS Load Balancer Operator 版本 0.0.1：

- RHEA-2022:5780 Release of AWS Load Balancer Operator on OperatorHub 增强公告更新

23.2. OPENSHIFT CONTAINER PLATFORM 中的 AWS LOAD BALANCER OPERATOR

AWS Load Balancer Operator 部署和管理 AWS Load Balancer Controller。您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装来自 OperatorHub 的 AWS Load Balancer Operator。

23.2.1. AWS Load Balancer Operator 的注意事项

在安装和使用 AWS Load Balancer Operator 前查看以下限制：

- IP 流量模式仅适用于 AWS Elastic Kubernetes Service (EKS)。AWS Load Balancer Operator 禁用 AWS Load Balancer Controller 的 IP 流量模式。禁用 IP 流量模式后，AWS Load Balancer Controller 无法使用 pod 就绪度。
- AWS Load Balancer Operator 将命令行标记（如 **--disable-ingress-class-annotation** 和 **--disable-ingress-group-name-annotation** ）添加到 AWS Load Balancer Controller。因此，AWS Load Balancer Operator 不允许在 **Ingress** 资源中使用 **kubernetes.io/ingress.class**

和 alb.ingress.kubernetes.io/group.name 注解。

23.2.2. AWS Load Balancer Operator

如果缺少 kubernetes.io/role/elb 标签，AWS Load Balancer Operator 可以标记公共子网。另外，AWS Load Balancer Operator 从底层 AWS 云检测到以下信息：

- 托管 Operator 的集群的虚拟私有云 (VPC) 的 ID。
- 发现 VPC 的公共和私有子网。

AWS Load Balancer Operator 支持类型为 LoadBalancer 的 Kubernetes 服务资源，使用只有 instance 目标类型的 Network Load Balancer (NLB)。

流程

1. 您可以通过运行以下命令来创建 Subscription 对象，以按需部署 AWS Load Balancer Operator：

```
$ oc -n aws-load-balancer-operator get sub aws-load-balancer-operator --
template='{{.status.installplan.name}}{{"\n"}}'
```

输出示例

```
install-zlfbt
```

2. 运行以下命令，检查安装计划的状态是否为 Complete ：

```
$ oc -n aws-load-balancer-operator get ip <install_plan_name> --
template='{{.status.phase}}{{"\n"}}'
```

输出示例

```
Complete
```

3. 运行以下命令，查看 aws-load-balancer-operator-controller-manager 部署的状态：

```
$ oc get -n aws-load-balancer-operator deployment/aws-load-balancer-operator-
controller-manager
```

输出示例

```
NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-operator-controller-manager  1/1     1            1           23h
```

23.2.3. 在 AWS VPC 集群中使用 AWS Load Balancer Operator 扩展至 Outpost

您可以配置 AWS Load Balancer Operator，以便在 AWS VPC 集群中置备 AWS Application Load Balancer。AWS Outposts 不支持 AWS Network Load Balancers。因此，AWS Load Balancer Operator 无法在 Outpost 中置备 Network Load Balancers。

您可以在云子网或 Outpost 子网中创建 AWS Application Load Balancer。云中的 Application Load Balancer 可以附加到基于云的计算节点，而 Outpost 中的 Application Load Balancer 可以附加到边缘计算节点。您必须使用 Outpost 子网或 VPC 子网来注解 Ingress 资源，但不能同时注解两者。

先决条件

- 您已将 AWS VPC 集群扩展到 Outpost。
- 已安装 OpenShift CLI（oc）。
- 已安装 AWS Load Balancer Operator 并创建了 AWS Load Balancer Controller。

流程

- 将 Ingress 资源配置为使用指定的子网：

  Ingress 资源配置示例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <application_name>
  annotations:
    alb.ingress.kubernetes.io/subnets: <subnet_id> 1
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <application_name>
                port:
                  number: 80
```

  1 指定要使用的子网。

  - 要在 Outpost 中使用 Application Load Balancer，请指定 Outpost 子网 ID。
  - 要在云中使用 Application Load Balancer，您必须在不同的可用区中指定至少两个子网。

23.2.4. AWS Load Balancer Operator 日志

您可以使用 oc logs 命令查看 AWS Load Balancer Operator 日志。

流程

- 运行以下命令，查看 AWS Load Balancer Operator 的日志：

```
$ oc logs -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager -c manager
```

-

23.3. 安装 AWS LOAD BALANCER OPERATOR

AWS Load Balancer Operator 部署和管理 AWS Load Balancer Controller。您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装来自 OperatorHub 的 AWS Load Balancer Operator。

23.3.1. 使用 Web 控制台安装 AWS Load Balancer Operator

您可以使用 Web 控制台安装 AWS Load Balancer Operator。

先决条件

- 已作为具有 **cluster-admin** 权限的用户身份登录 OpenShift Container Platform Web 控制台。
- 集群被配置为使用 AWS 作为平台类型和云供应商。
- 如果您使用安全令牌服务(STS)或用户置备的基础架构，请按照相关的准备步骤操作。例如，如果您使用 AWS 安全令牌服务，请参阅使用 AWS 安全令牌服务(STS) "在集群中准备 AWS Load Balancer Operator"。

流程

1. 在 OpenShift Container Platform Web 控制台中进入 Operators → OperatorHub。
2. 选择 AWS Load Balancer Operator。您可以使用 Filter by keyword 文本框，或者使用过滤器列表从 Operator 列表搜索 AWS Load Balancer Operator。
3. 选择 **aws-load-balancer-operator** 命名空间。
4. 在 Install Operator 页面中，选择以下选项：
   a. 更新频道为 stable-v1。
   b. 安装模式 为 All namespaces on the cluster (default)。
   c. Installed Namespace 为 **aws-load-balancer-operator**。如果 **aws-load-balancer-operator** 命名空间不存在，它会在 Operator 安装过程中创建。
   d. 选择 Update approval 为 Automatic 或 Manual。默认情况下，Update approval 设置为 Automatic。如果选择自动更新，Operator Lifecycle Manager(OLM)将自动升级 Operator 的运行实例，而无需任何干预。如果选择手动更新，OLM 将创建一个更新请求。作为集群管理员，您必须手动批准该更新请求，以便将 Operator 更新至新版本。
5. 点 Install。

验证

- 在 Installed Operators 仪表板中验证 AWS Load Balancer Operator 的 Status 显示为 Succeeded。

23.3.2. 使用 CLI 安装 AWS Load Balancer Operator

您可以使用 CLI 安装 AWS Load Balancer Operator。

先决条件

- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform Web 控制台。
- 集群被配置为使用 AWS 作为平台类型和云供应商。
- 已登陆到 OpenShift CLI (oc)。

流程

1. 创建一个 Namespace 对象：
   a. 创建定义 Namespace 对象的 YAML 文件：

      namespace.yaml 文件示例

      ```
      apiVersion: v1
      kind: Namespace
      metadata:
        name: aws-load-balancer-operator
      ```

   b. 运行以下命令来创建 Namespace 对象：

      ```
      $ oc apply -f namespace.yaml
      ```

2. 创建一个 OperatorGroup 对象：
   a. 创建定义 OperatorGroup 对象的 YAML 文件：

      operatorgroup.yaml 文件示例

      ```
      apiVersion: operators.coreos.com/v1
      kind: OperatorGroup
      metadata:
        name: aws-lb-operatorgroup
        namespace: aws-load-balancer-operator
      spec:
        upgradeStrategy: Default
      ```

   b. 运行以下命令来创建 OperatorGroup 对象：

      ```
      $ oc apply -f operatorgroup.yaml
      ```

3. 创建 Subscription 对象：
   a. 创建定义 Subscription 对象的 YAML 文件：

      subscription.yaml 文件示例

      ```
      apiVersion: operators.coreos.com/v1alpha1
      kind: Subscription
      metadata:
        name: aws-load-balancer-operator
        namespace: aws-load-balancer-operator
      spec:
        channel: stable-v1
      ```

```
installPlanApproval: Automatic
name: aws-load-balancer-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
```

b. 运行以下命令来创建 Subscription 对象：

```
$ oc apply -f subscription.yaml
```

验证

1. 从订阅中获取安装计划的名称：

```
$ oc -n aws-load-balancer-operator \
    get subscription aws-load-balancer-operator \
    --template='{{.status.installplan.name}}{{"\n"}}'
```

2. 检查安装计划的状态：

```
$ oc -n aws-load-balancer-operator \
    get ip <install_plan_name> \
    --template='{{.status.phase}}{{"\n"}}'
```

输出必须是 Complete。

23.4. 使用 AWS 安全令牌服务在集群中准备 AWS LOAD BALANCER OPERATOR

您可以在使用 STS 的集群中安装 AWS Load Balancer Operator。在安装 Operator 前，按照以下步骤准备集群。

AWS Load Balancer Operator 依赖于 CredentialsRequest 对象来引导 Operator 和 AWS Load Balancer Controller。AWS Load Balancer Operator 等待所需的 secret 创建并可用。

23.4.1. 为 AWS Load Balancer Operator 创建 IAM 角色

需要额外的 AWS Identity and Access Management (IAM) 角色，才能在使用 STS 的集群中安装 AWS Load Balancer Operator。需要 IAM 角色与子网和虚拟私有云(VPC)交互。AWS Load Balancer Operator 使用 IAM 角色生成 CredentialsRequest 对象来引导其自身。

您可以使用以下选项创建 IAM 角色：

- 使用 Cloud Credential Operator 实用程序(ccoctl) 和预定义的 CredentialsRequest 对象。
- 使用 AWS CLI 和预定义的 AWS 清单。

如果您的环境不支持 ccoctl 命令，请使用 AWS CLI。

23.4.1.1. 使用 Cloud Credential Operator 实用程序创建 AWS IAM 角色

您可以使用 Cloud Credential Operator 实用程序(ccoctl) 为 AWS Load Balancer Operator 创建 AWS IAM 角色。AWS IAM 角色用于与子网和虚拟私有云 (VPC) 交互。

先决条件

- 您必须提取并准备 ccoctl 二进制文件。

流程

1. 运行以下命令，下载 CredentialsRequest 自定义资源 (CR) 并将其存储在目录中：

```
$ curl --create-dirs -o <credrequests-dir>/operator.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/hack/operator-credentials-request.yaml
```

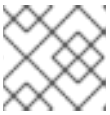
2. 运行以下命令，使用 ccoctl 实用程序创建 AWS IAM 角色：

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credrequests-dir> \
    --identity-provider-arn <oidc-arn>
```

输出示例

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-
operator-aws-load-balancer-operator created (1)
2023/09/12 11:38:57 Saved credentials configuration to: /home/user/<credrequests-
dir>/manifests/aws-load-balancer-operator-aws-load-balancer-operator-
credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-
aws-load-balancer-operator created
```

(1) 请注意 AWS IAM 角色的 Amazon 资源名称(ARN)。



注意

AWS IAM 角色名称的长度必须小于或等于 12 个字符。

23.4.1.2. 使用 AWS CLI 创建 AWS IAM 角色

您可以使用 AWS 命令行界面为 AWS Load Balancer Operator 创建 IAM 角色。IAM 角色用于与子网和虚拟私有云 (VPC) 交互。

先决条件

- 您必须有权访问 AWS 命令行界面 (aws)。

流程

1. 运行以下命令，使用身份提供程序生成信任策略文件：

```
$ cat <<EOF > albo-operator-trust-policy.json
{
    "Version": "2012-10-17",
```

```
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Federated": "arn:aws:iam::777777777777:oidc-provider/<oidc-provider-id>"
(1)
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
          "StringEquals": {
            "<oidc-provider-id>:sub": "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-operator-controller-manager" (2)
          }
        }
      }
    ]
}
EOF
```

(1) 指定身份提供程序的 Amazon 资源名称 (ARN)。

(2) 指定 AWS Load Balancer Operator 的服务帐户。

2. 运行以下命令，使用生成的信任策略创建 IAM 角色：

```
$ aws iam create-role --role-name albo-operator --assume-role-policy-document
file://albo-operator-trust-policy.json
```

输出示例

```
ROLE arn:aws:iam::777777777777:role/albo-operator 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-
balancer-controller-manager
PRINCIPAL arn:aws:iam:777777777777:oidc-provider/<oidc-provider-id>
```

(1) 记录创建的 IAM 角色的 ARN。

3. 运行以下命令，下载 AWS Load Balancer Operator 的权限策略：

```
$ curl -o albo-operator-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/hack/operator-permission-policy.json
```

4. 运行以下命令，将 AWS Load Balancer Controller 的权限策略附加到 IAM 角色：

```
$ aws iam put-role-policy --role-name albo-operator --policy-name perms-policy-albo-
operator --policy-document file://albo-operator-permission-policy.json
```

23.4.2. 为 AWS Load Balancer Operator 配置 ARN 角色

您可以将 AWS Load Balancer Operator 的 Amazon 资源名称 (ARN) 角色配置为环境变量。您可以使用 CLI 配置 ARN 角色。

先决条件

- 已安装 OpenShift CLI（oc）。

流程

1. 运行以下命令来创建 aws-load-balancer-operator 项目：

```
$ oc new-project aws-load-balancer-operator
```

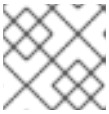
2. 运行以下命令来创建 OperatorGroup 对象：

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  targetNamespaces: []
EOF
```

3. 运行以下命令来创建 Subscription 对象：

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: ROLEARN
      value: "<role-arn>" (1)
EOF
```

(1) 指定 CredentialsRequest 中使用的 ARN 角色，以便为 AWS Load Balancer Operator 置备 AWS 凭证。



注意

AWS Load Balancer Operator 会在进入 Available 状态前等待创建 secret。

23.4.3. 为 AWS Load Balancer Controller 创建 IAM 角色

AWS Load Balancer Controller 的 **CredentialsRequest** 对象必须使用手动置备的 IAM 角色设置。

您可以使用以下选项创建 IAM 角色：

- 使用 Cloud Credential Operator 实用程序(**ccoctl**) 和预定义的 **CredentialsRequest** 对象。
- 使用 AWS CLI 和预定义的 AWS 清单。

如果您的环境不支持 **ccoctl** 命令，请使用 AWS CLI。

23.4.3.1. 使用 Cloud Credential Operator 实用程序为控制器创建 AWS IAM 角色

您可以使用 Cloud Credential Operator 实用程序(**ccoctl**) 为 AWS Load Balancer Controller 创建 AWS IAM 角色。AWS IAM 角色用于与子网和虚拟私有云 (VPC) 交互。

先决条件

- 您必须提取并准备 **ccoctl** 二进制文件。

流程

1. 运行以下命令，下载 **CredentialsRequest** 自定义资源 (CR) 并将其存储在目录中：

```
$ curl --create-dirs -o <credrequests-dir>/controller.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/hack/controller/controller-credentials-request.yaml
```

2. 运行以下命令，使用 **ccoctl** 实用程序创建 AWS IAM 角色：

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credrequests-dir> \
    --identity-provider-arn <oidc-arn>
```

输出示例

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-
operator-aws-load-balancer-controller created (1)
2023/09/12 11:38:57 Saved credentials configuration to: /home/user/<credrequests-
dir>/manifests/aws-load-balancer-operator-aws-load-balancer-controller-
credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-
aws-load-balancer-controller created
```

(1) 请注意 AWS IAM 角色的 Amazon 资源名称(ARN)。

注意

AWS IAM 角色名称的长度必须小于或等于 12 个字符。

23.4.3.2. 使用 AWS CLI 为控制器创建 AWS IAM 角色

您可以使用 AWS 命令行界面为 AWS Load Balancer Controller 创建 AWS IAM 角色。AWS IAM 角色用于与子网和虚拟私有云 (VPC) 交互。

先决条件

- 您必须有权访问 AWS 命令行界面 (aws)。

流程

1. 运行以下命令，使用身份提供程序生成信任策略文件：

```
$ cat <<EOF > albo-controller-trust-policy.json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Federated": "arn:aws:iam::777777777777:oidc-provider/<oidc-provider-id>"
(1)
         },
         "Action": "sts:AssumeRoleWithWebIdentity",
         "Condition": {
            "StringEquals": {
               "<oidc-provider-id>:sub": "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-controller-cluster" (2)
            }
         }
      }
   ]
}
EOF
```

(1) 指定身份提供程序的 Amazon 资源名称 (ARN)。

(2) 指定 AWS Load Balancer Controller 的服务帐户。

2. 运行以下命令，使用生成的信任策略创建 AWS IAM 角色：

```
$ aws iam create-role --role-name albo-controller --assume-role-policy-document
file://albo-controller-trust-policy.json
```

输出示例

```
ROLE arn:aws:iam::777777777777:role/albo-controller 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-
balancer-controller-cluster
PRINCIPAL arn:aws:iam:777777777777:oidc-provider/<oidc-provider-id>
```

(1) 记录 AWS IAM 角色的 ARN。

3. 运行以下命令，下载 AWS Load Balancer Controller 的权限策略：

```
$ curl -o albo-controller-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/assets/iam-policy.json
```

4. 运行以下命令，将 AWS Load Balancer Controller 的权限策略附加到 AWS IAM 角色：

```
$ aws iam put-role-policy --role-name albo-controller --policy-name perms-policy-albo-
controller --policy-document file://albo-controller-permission-policy.json
```

5. 创建定义 **AWSLoadBalancerController** 对象的 YAML 文件：

sample-aws-lb-manual-creds.yaml 文件示例：

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController (1)
metadata:
  name: cluster (2)
spec:
  credentialsRequestConfig:
    stsIAMRoleARN: <role-arn> (3)
```

(1) 定义 **AWSLoadBalancerController** 对象。

(2) 定义 AWS Load Balancer Controller 名称。所有相关资源都使用此实例名称作为后缀。

(3) 指定 ARN 角色。**CredentialsRequest** 对象使用此 ARN 角色来置备 AWS 凭证。

23.4.4. 其他资源

- 配置 Cloud Credential Operator 工具

23.5. 创建 AWS LOAD BALANCER CONTROLLER 实例

安装 AWS Load Balancer Operator 后，您可以创建 AWS Load Balancer Controller。

23.5.1. 创建 AWS Load Balancer Controller

您只能在集群中安装 **AWSLoadBalancerController** 对象的单个实例。您可以使用 CLI 创建 AWS Load Balancer Controller。AWS Load Balancer Operator 只协调名为 resource 的集群。

先决条件

- 您已创建了 **echoserver** 命名空间。
- 您可以访问 OpenShift CLI(**oc**)。

流程

1. 创建定义 **AWSLoadBalancerController** 对象的 YAML 文件：

sample-aws-lb.yaml 文件示例

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController (1)
metadata:
  name: cluster (2)
spec:
  subnetTagging: Auto (3)
  additionalResourceTags: (4)
  - key: example.org/security-scope
    value: staging
  ingressClass: alb (5)
  config:
    replicas: 2 (6)
  enabledAddons: (7)
    - AWSWAFv2 (8)
```

(1) 定义 AWSLoadBalancerController 对象。

(2) 定义 AWS Load Balancer Controller 名称。此实例名称作为后缀添加到所有相关资源。

(3) 配置 AWS Load Balancer Controller 的子网标记方法。以下值有效：

- Auto ：AWS Load Balancer Operator 决定属于集群的子网，并相应地标记它们。如果内部子网上不存在内部子网标签，Operator 无法正确确定角色。
- Manual ：您可以使用适当的角色标签手动标记属于集群的子网。如果在用户提供的基础架构上安装集群，则使用这个选项。

(4) 定义在置备 AWS 资源时 AWS Load Balancer Controller 使用的标签。

(5) 定义入口类名称。默认值为 alb。

(6) 指定 AWS Load Balancer Controller 的副本数。

(7) 将注解指定为 AWS Load Balancer Controller 的附加组件。

(8) 启用 alb.ingress.kubernetes.io/wafv2-acl-arn 注解。

2. 运行以下命令来创建 AWSLoadBalancerController 对象：

```
$ oc create -f sample-aws-lb.yaml
```

3. 创建定义 Deployment 资源的 YAML 文件：

sample-aws-lb.yaml 文件示例

```
apiVersion: apps/v1
kind: Deployment (1)
metadata:
  name: <echoserver> (2)
  namespace: echoserver
spec:
```

```
  selector:
    matchLabels:
      app: echoserver
  replicas: 3 3
  template:
    metadata:
      labels:
        app: echoserver
    spec:
      containers:
        - image: openshift/origin-node
          command:
           - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"'
          imagePullPolicy: Always
          name: echoserver
          ports:
            - containerPort: 8080
```

1. 定义部署资源。
2. 指定部署名称。
3. 指定部署的副本数量。

4. 创建定义 Service 资源的 YAML 文件：

   service-albo.yaml 文件示例：

```
apiVersion: v1
kind: Service 1
metadata:
  name: <echoserver> 2
  namespace: echoserver
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector:
    app: echoserver
```

1. 定义服务资源。
2. 指定服务名称。

5. 创建定义 Ingress 资源的 YAML 文件：

   ingress-albo.yaml 文件示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <name> 1
  namespace: echoserver
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <echoserver> 2
                port:
                  number: 80
```

1 为 Ingress 资源指定一个名称。

2 指定服务名称。

验证

- 运行以下命令，将 Ingress 资源的状态保存到 HOST 变量中：

```
$ HOST=$(oc get ingress -n echoserver echoserver --template='{{(index
.status.loadBalancer.ingress 0).hostname}}')
```

- 运行以下命令，验证 Ingress 资源的状态：

```
$ curl $HOST
```

23.6. 通过单个 AWS LOAD BALANCER 提供多个入口资源

您可以通过单个 AWS Load Balancer 将流量路由到属于单个域的不同服务。每个 Ingress 资源提供域的不同端点。

23.6.1. 通过单个 AWS Load Balancer 创建多个入口资源

您可以使用 CLI 通过单个 AWS Load Balancer 将流量路由到多个入口资源。

先决条件

- 您可以访问 OpenShift CLI(oc)。

流程

1. 创建一个 IngressClassParams 资源 YAML 文件，如sample-single-lb-params.yaml，如下所示：

```
apiVersion: elbv2.k8s.aws/v1beta1 1
kind: IngressClassParams
metadata:
  name: single-lb-params 2
spec:
  group:
    name: single-lb 3
```

1. 定义 IngressClassParams 资源的 API 组和版本。
2. 指定 IngressClassParams 资源名称。
3. 指定 IngressGroup 资源名称。此类的所有 Ingress 资源都属于此 IngressGroup。

2. 运行以下命令来创建 IngressClassParams 资源：

```
$ oc create -f sample-single-lb-params.yaml
```

3. 创建 IngressClass 资源 YAML 文件，如sample-single-lb-class.yaml，如下所示：

```
apiVersion: networking.k8s.io/v1 1
kind: IngressClass
metadata:
  name: single-lb 2
spec:
  controller: ingress.k8s.aws/alb 3
  parameters:
    apiGroup: elbv2.k8s.aws 4
    kind: IngressClassParams 5
    name: single-lb-params 6
```

1. 定义 IngressClass 资源的 API 组和版本。
2. 指定入口类名称。
3. 定义控制器名称。ingress.k8s.aws/alb 值表示此类的所有入口资源都应由 AWS Load Balancer Controller 管理。
4. 定义 IngressClassParams 资源的 API 组。
5. 定义 IngressClassParams 资源的资源类型。
6. 定义 IngressClassParams 资源名称。

4. 运行以下命令来创建 IngressClass 资源：

```
$ oc create -f sample-single-lb-class.yaml
```

5. 创建 AWSLoadBalancerController 资源 YAML 文件，如sample-single-lb.yaml，如下所示：

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: single-lb 1
```

1 定义 IngressClass 资源的名称。

6. 运行以下命令来创建 AWSLoadBalancerController 资源：

```
$ oc create -f sample-single-lb.yaml
```

7. 创建 Ingress 资源 YAML 文件，如sample-multiple-ingress.yaml，如下所示：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-1 1
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing 2
    alb.ingress.kubernetes.io/group.order: "1" 3
    alb.ingress.kubernetes.io/target-type: instance 4
spec:
  ingressClassName: single-lb 5
  rules:
  - host: example.com 6
    http:
      paths:
      - path: /blog 7
        pathType: Prefix
        backend:
          service:
            name: example-1 8
            port:
              number: 80 9
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-2
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "2"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
      paths:
      - path: /store
```

```
        pathType: Prefix
        backend:
          service:
            name: example-2
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-3
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "3"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: example-3
            port:
              number: 80
```

1. 指定入口名称。
2. 指明在公共子网中置备的负载均衡器，以访问互联网。
3. 指定在负载均衡器收到请求时，来自多个入口资源的规则的顺序匹配。
4. 表示负载均衡器将针对 OpenShift Container Platform 节点为目标，以访问该服务。
5. 指定属于此入口的入口类。
6. 定义用于请求路由的域名。
7. 定义必须路由到服务的路径。
8. 定义提供 Ingress 资源中配置的端点的服务名称。
9. 定义服务上提供端点的端口。

8. 运行以下命令来创建 Ingress 资源：

```
$ oc create -f sample-multiple-ingress.yaml
```

23.7. 添加 TLS 终止

您可以在 AWS Load Balancer 上添加 TLS 终止。

23.7.1. 在 AWS Load Balancer 中添加 TLS 终止

您可以将域的流量路由到服务的 pod，并在 AWS 负载均衡器中添加 TLS 终止。

先决条件

- 您可以访问 OpenShift CLI(oc)。

流程

1. 创建定义 **AWSLoadBalancerController** 资源的 YAML 文件：

   add-tls-termination-albc.yaml 文件示例

   ```
   apiVersion: networking.olm.openshift.io/v1
   kind: AWSLoadBalancerController
   metadata:
     name: cluster
   spec:
     subnetTagging: Auto
     ingressClass: tls-termination 1
   ```

   1 定义入口类名称。如果集群中没有 ingress 类，AWS Load Balancer Controller 会创建一个。如果 **spec.controller** 设置为 **ingress.k8s.aws/alb**，AWS Load Balancer Controller 会协调额外的入口类值。

2. 创建定义 **Ingress** 资源的 YAML 文件：

   add-tls-termination-ingress.yaml 文件示例

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: <example> 1
     annotations:
       alb.ingress.kubernetes.io/scheme: internet-facing 2
       alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:xxxxx 3
   spec:
     ingressClassName: tls-termination 4
     rules:
     - host: <example.com> 5
       http:
         paths:
           - path: /
             pathType: Exact
             backend:
               service:
                 name: <example-service> 6
                 port:
                   number: 80
   ```

   1 指定入口名称。

2 控制器在公共子网中为入口置备负载均衡器，以便通过互联网访问负载均衡器。

3 附加到负载均衡器的证书的 Amazon 资源名称(ARN)。

4 定义入口类名称。

5 定义流量路由的域。

6 定义流量路由的服务。

23.8. 配置集群范围代理

您可以在 AWS Load Balancer Operator 中配置集群范围代理。配置集群范围代理后，Operator Lifecycle Manager (OLM) 会使用 HTTP_PROXY、HTTPS_PROXY 和 NO_PROXY 等环境变量自动更新 Operator 的所有部署。这些变量由 AWS Load Balancer Operator 填充给受管控制器。

23.8.1. 信任集群范围代理的证书颁发机构

1. 运行以下命令，创建配置映射以在 aws-load-balancer-operator 命名空间中包含证书颁发机构 (CA) 捆绑包：

```
$ oc -n aws-load-balancer-operator create configmap trusted-ca
```

2. 要将可信 CA 捆绑包注入配置映射中，请运行以下命令将 config.openshift.io/inject-trusted-cabundle=true 标签添加到配置映射中：

```
$ oc -n aws-load-balancer-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

3. 运行以下命令，更新 AWS Load Balancer Operator 订阅以访问 AWS Load Balancer Operator 部署中的配置映射：

```
$ oc -n aws-load-balancer-operator patch subscription aws-load-balancer-operator --type='merge' -p '{"spec":{"config":{"env":[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}],"volumes":[{"name":"trusted-ca","configMap":{"name":"trusted-ca"}}],"volumeMounts":[{"name":"trusted-ca","mountPath":"/etc/pki/tls/certs/albo-tls-ca-bundle.crt","subPath":"ca-bundle.crt"}]}}}'
```

4. 部署 AWS Load Balancer Operator 后，运行以下命令来验证 CA 捆绑包是否已添加到 aws-load-balancer-operator-controller-manager 部署中：

```
$ oc -n aws-load-balancer-operator exec deploy/aws-load-balancer-operator-controller-manager -c manager -- bash -c "ls -l /etc/pki/tls/certs/albo-tls-ca-bundle.crt; printenv TRUSTED_CA_CONFIGMAP_NAME"
```

输出示例

```
-rw-r--r--. 1 root 1000690000 5875 Jan 11 12:25 /etc/pki/tls/certs/albo-tls-ca-bundle.crt
trusted-ca
```

5. **可选：通过运行以下命令，每次 configmap 发生变化时重启 AWS Load Balancer Operator 的部署：**

```
$ oc -n aws-load-balancer-operator rollout restart deployment/aws-load-balancer-operator-controller-manager
```

23.8.2. 其他资源

- 使用 Operator 进行证书注入

第 24 章 多网络

24.1. 了解多网络

在 Kubernetes 中，容器网络被委派给实现 Container Network Interface (CNI) 的网络插件。

OpenShift Container Platform 使用 Multus CNI 插件来串联 CNI 插件。在集群安装过程中，您要配置 *default* pod 网络。默认网络处理集群中的所有一般网络流量。您可以基于可用的 CNI 插件定义*额外网络*，并将一个或多个此类网络附加到 pod。您可以根据需要为集群定义多个额外网络。这可让您灵活地配置提供交换或路由等网络功能的 pod。

24.1.1. 额外网络使用场景

您可以在需要网络隔离的情况下使用额外网络，包括分离数据平面与控制平面。隔离网络流量对以下性能和安全性原因很有用：

性能

您可以在两个不同的平面上发送流量，以管理每个平面上流量的多少。

安全性

您可以将敏感的流量发送到专为安全考虑而管理的网络平面，也可隔离不能在租户或客户间共享的私密数据。

集群中的所有 pod 仍然使用集群范围的默认网络，以维持整个集群中的连通性。每个 pod 都有一个 **eth0** 接口，附加到集群范围的 pod 网络。您可以使用 **oc exec -it <pod_name> -- ip a** 命令来查看 pod 的接口。如果您添加使用 Multus CNI 的额外网络接口，则名称为 **net1**、**net2**、...、**netN**。

要将额外网络接口附加到 pod，您必须创建配置来定义接口的附加方式。您可以使用 **NetworkAttachmentDefinition** 自定义资源（CR）来指定各个接口。各个 CR 中的 CNI 配置定义如何创建该接口。

24.1.2. OpenShift Container Platform 中的额外网络

OpenShift Container Platform 提供以下 CNI 插件，以便在集群中创建额外网络：

- bridge：配置基于网桥的额外网络，以允许同一主机上的 pod 相互通信，并与主机通信。
- host-device：配置 host-device 额外网络，以允许 pod 访问主机系统上的物理以太网网络设备。
- ipvlan：配置基于 ipvlan 的额外网络，以允许主机上的 Pod 与其他主机和那些主机上的 pod 通信，这类似于基于 macvlan 的额外网络。与基于 macvlan 的额外网络不同，每个 pod 共享与父级物理网络接口相同的 MAC 地址。
- vlan：配置基于 vlan 的额外网络，以为 pod 启用基于 VLAN 的网络隔离和连接。
- macvlan：配置基于 macvlan 的额外网络，以允许主机上的 Pod 通过使用物理网络接口与其他主机和那些主机上的 Pod 通信。附加到基于 macvlan 的额外网络的每个 pod 都会获得一个唯一的 MAC 地址。
- tap：配置基于 tap 的额外网络，以在容器命名空间内创建 tap 设备。tap 设备可让用户空间程序发送和接收网络数据包。
- SR-IOV：配置基于 SR-IOV 的额外网络，以允许 pod 附加到主机系统上支持 SR-IOV 的硬件的虚拟功能(VF)接口。

24.2. 配置额外网络

作为集群管理员，您可以为集群配置额外网络。支持以下网络类型：

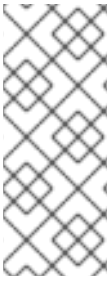
- Bridge
- 主机设备
- VLAN
- IPVLAN
- MACVLAN
- TAP
- OVN-Kubernetes

24.2.1. 管理额外网络的方法

您可以通过两种方法来管理额外网络的生命周期。每种方法都是相互排斥的，您一次只能使用一种方法来管理额外网络。对于任一方法，额外网络由您配置的 Container Network Interface(CNI)插件管理。

对于额外网络，IP 地址通过您配置为额外网络一部分的 IP 地址管理 (IPAM) CNI 插件来置备。IPAM 插件支持多种 IP 地址分配方法，包括 DHCP 和静态分配。

- 修改 Cluster Network Operator(CNO)配置：CNO 会自动创建和管理 **NetworkAttachmentDefinition** 对象。除了管理对象生命周期外，CNO 可以确保 DHCP 可用于使用 DHCP 分配的 IP 地址的额外网络。
- 应用 YAML 清单：您可以通过创建 **NetworkAttachmentDefinition** 对象直接管理额外网络。这个方法可以串联 CNI 插件。



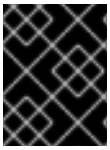
注意

当使用 OVN SDN 在 Red Hat OpenStack Platform (RHOSP) 中使用多个网络接口部署 OpenShift Container Platform 节点时，二级接口的 DNS 配置可能会优先于主接口的 DNS 配置。在这种情况下，删除附加到二级接口的子网 ID 的 DNS 名称服务器：

```
$ openstack subnet set --dns-nameserver 0.0.0.0 <subnet_id>
```

24.2.2. 配置额外网络附加

额外网络通过使用 **k8s.cni.cncf.io** API 组中的 **NetworkAttachmentDefinition** API 来配置。



重要

请勿将任何敏感信息或机密存储在 **NetworkAttachmentDefinition** 对象中，因为此类信息可由项目管理用户访问。

下表中描述了 API 的配置：

表 24.1. **NetworkAttachmentDefinition** API 字段

字段	类型	描述
metadata.name	string	额外网络的名称。
metadata.namespace	string	与对象关联的命名空间。
spec.config	string	JSON 格式的 CNI 插件配置。

24.2.2.1. 通过 Cluster Network Operator 配置额外网络

额外网络附加的配置作为 Cluster Network Operator(CNO)配置的一部分被指定。

以下 YAML 描述了使用 CNO 管理额外网络的配置参数：

Cluster Network Operator 配置

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks: 1
  - name: <name> 2
    namespace: <namespace> 3
    rawCNIConfig: |- 4
      {
        ...
      }
    type: Raw
```

1 由一个或多个附加网络配置组成的数组。

2 您要创建的额外网络附加的名称。该名称在指定的 namespace 中需要是唯一的。

3 在其中创建网络附加的命名空间。如果您未指定值，则使用 default 命名空间。

4 JSON 格式的 CNI 插件配置。

24.2.2.2. 从 YAML 清单配置额外网络

从 YAML 配置文件指定额外网络的配置，如下例所示：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
spec:
  config: |- 2
```

```
{
  ...
}
```

1. 您要创建的额外网络附加的名称。
2. JSON 格式的 CNI 插件配置。

24.2.3. 额外网络类型的配置

以下部分介绍了额外网络的具体配置字段。

24.2.3.1. 配置桥接额外网络

以下对象描述了 Bridge CNI 插件的配置参数：

表 24.2. bridge CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	用于配置的 CNI 插件的名称：**bridge**。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
bridge	**string**	可选：指定要使用的虚拟网桥名称。如果主机上不存在网桥接口，则进行创建。默认值为 **cni0**。
ipMasq	**布尔值**	可选：设置为 **true**，为离开虚拟网络的流量启用 IP 伪装。所有流量的源 IP 地址都会改写为网桥 IP 地址。如果网桥没有 IP 地址，此设置无效。默认值为 **false**。
isGateway	**布尔值**	可选：设置为 **true**，从而为网桥分配 IP 地址。默认值为 **false**。
isDefaultGateway	**布尔值**	可选：设置为 **true**，将网桥配置为虚拟网络的默认网关。默认值为 **false**。如果 **isDefaultGateway** 设置为 **true**，则 **isGateway** 也会自动设置为 **true**。
forceAddress	**布尔值**	可选：设置为 **true**，以允许将之前分配的 IP 地址分配给虚拟网桥。设置为 **false** 时，如果将来自于重叠子集的 IPv4 地址或者 IPv6 地址分配给虚拟网桥，则会发生错误。默认值为 **false**。
hairpinMode	**布尔值**	可选：设置为 **true**，以允许虚拟网桥通过收到它的虚拟端口将其发回。这个模式也被称为*反射中继*。默认值为 **false**。
promiscMode	**布尔值**	可选：设置为 **true** 以在网桥上启用混杂模式。默认值为 **false**。

字段	类型	描述
vlan	**string**	可选：指定一个虚拟 LAN (VLAN) 标签作为整数值。默认情况下不分配 VLAN 标签。
preserveDefaultVlan	**string**	可选：指示在连接到网桥的 **veth** 端是否保留默认 vlan。默认值为 true。
vlanTrunk	**list**	可选：分配 VLAN 中继标签。默认值为 **none**。
mtu	**string**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
enabledad	**布尔值**	可选：为容器侧 **veth** 启用重复的地址检测。默认值为 **false**。
macspoofchk	**布尔值**	可选：启用 mac spoof 检查，将来自容器的流量限制为接口的 mac 地址。默认值为 **false**。

注意

VLAN 参数在 veth 的主机端配置 VLAN 标签，并在网桥接口上启用 vlan_filtering 功能。

注意

要为 L2 网络配置 uplink，您必须使用以下命令在 uplink 接口上允许 VLAN：

```
$ bridge vlan add vid VLAN_ID dev DEV
```

24.2.3.1.1. Bridge CNI 插件配置示例

以下示例配置了名为 bridge-net 的额外网络：

```
{
  "cniVersion": "0.3.1",
  "name": "bridge-net",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}
```

24.2.3.2. 主机设备额外网络配置

注意

仅设置以下参数之一来指定您的网络设备：device、hwaddr、kernelpath 或 pciBusID。

以下对象描述了 host-device CNI 插件的配置参数：

表 24.3. 主机 device CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	用于配置的 CNI 插件的名称：**host-device**。
device	**string**	可选：设备的名称，如 **eth0**。
hwaddr	**string**	可选：设备硬件 MAC 地址。
kernelpath	**string**	可选：Linux 内核设备路径，如 **/sys/devices/pci0000:00/0000:00:1f.6**。
pciBusID	**string**	可选：网络设备的 PCI 地址，如 **0000:00:1f.6**。

24.2.3.2.1. host-device 配置示例

以下示例配置了名为 **hostdev-net** 的额外网络：

```
{
  "cniVersion": "0.3.1",
  "name": "hostdev-net",
  "type": "host-device",
  "device": "eth1"
}
```

24.2.3.3. 配置 VLAN 额外网络

以下对象描述了 VLAN CNI 插件的配置参数：

表 24.4. VLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**vlan**。
master	**string**	与网络附加关联的以太网接口。如果没有指定 **master**，则使用默认网络路由的接口。

字段	类型	描述
vlanId	**整数**	设置 vlan 的 id。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
dns	**整数**	可选：返回的 DNS 信息，例如，用于优先级排序的 DNS 名称服务器列表。
linkInContainer	**布尔值**	可选：指定 master 接口是否在容器网络命名空间或主网络命名空间中。将值设为 **true** 以请求使用容器命名空间 master 接口。

24.2.3.3.1. VLAN 配置示例

以下示例配置了一个名为 vlan-net 的额外网络：

```
{
  "name": "vlan-net",
  "cniVersion": "0.3.1",
  "type": "vlan",
  "master": "eth0",
  "mtu": 1500,
  "vlanId": 5,
  "linkInContainer": false,
  "ipam": {
     "type": "host-local",
     "subnet": "10.1.1.0/24"
  },
  "dns": {
     "nameservers": [ "10.1.1.1", "8.8.8.8" ]
  }
}
```

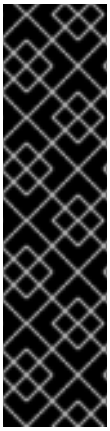
24.2.3.4. 配置 ipvlan 额外网络

以下对象描述了 IPVLAN CNI 插件的配置参数：

表 24.5. IPVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**ipvlan**。

字段	类型	描述
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。除非插件被串联，否则需要此项。
模式	**string**	可选：虚拟网络的操作模式。这个值必须是 **l2**、**l3** 或 **l3s**。默认值为 **l2**。
master	**string**	可选：与网络附加关联的以太网接口。如果没有指定 **master**，则使用默认网络路由的接口。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
linkInContainer	**布尔值**	可选：指定 master 接口是否在容器网络命名空间或主网络命名空间中。将值设为 **true** 以请求使用容器命名空间 master 接口。



重要

- **ipvlan 对象不允许虚拟接口与 master 接口通信。因此，容器无法使用 ipvlan 接口来访问主机。确保容器加入提供主机连接的网络，如支持 Precision Time Protocol (PTP) 的网络。**
- **单个 master 接口无法同时配置为使用 macvlan 和 ipvlan。**
- **对于不能与接口无关的 IP 分配方案，可以使用处理此逻辑的较早插件来串联 ipvlan 插件。如果省略 master，则前面的结果必须包含一个接口名称，以便 ipvlan 插件进行 enslave。如果省略 ipam，则使用前面的结果来配置 ipvlan 接口。**

24.2.3.4.1. IPVLAN CNI 插件配置示例

以下示例配置了名为 ipvlan -net 的额外网络：

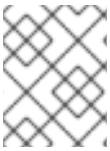
```
{
  "cniVersion": "0.3.1",
  "name": "ipvlan-net",
  "type": "ipvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

24.2.3.5. 配置 macvlan 额外网络

以下对象描述了 MACVLAN CNI 插件的配置参数：

表 24.6. MACVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	用于配置的 CNI 插件的名称：**macvlan**。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
模式	**string**	可选：配置虚拟网络上的流量可见性。必须是 **bridge**、**passthru**、**private**或 **Vepa**。如果没有提供值，则默认值为 **bridge**。
master	**string**	可选：与新创建的 macvlan 接口关联的主机网络接口。如果没有指定值，则使用默认路由接口。
mtu	**string**	可选：将最大传输单元 (MTU) 到指定的值。默认值由内核自动设置。
linkInContainer	**布尔值**	可选：指定 master 接口是否在容器网络命名空间或主网络命名空间中。将值设为 **true** 以请求使用容器命名空间 master 接口。



注意

如果您为插件配置指定 master key，请使用与主网络插件关联的物理网络接口，以避免可能冲突。

24.2.3.5.1. MACVLAN CNI 插件配置示例

以下示例配置了名为 **macvlan-net** 的额外网络：

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
    }
}
```

24.2.3.6. 配置 TAP 额外网络

以下对象描述了TAP CNI 插件的配置参数：

表 24.7. TAP CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**tap**。
mac	**string**	可选：为接口请求指定的 MAC 地址。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
selinuxcontext	**string**	可选：与 tap 设备关联的 SELinux 上下文。  **注意** OpenShift Container Platform 需要 **system_u:system_r:container_t:s0** 的值。
multiQueue	**布尔值**	可选：设置为 **true** 以启用多队列。
owner	**整数**	可选：拥有 tap 设备的用户。
group	**整数**	可选：拥有 tap 设备的组。
bridge	**string**	可选：将 tap 设备设置为已存在的网桥的端口。

24.2.3.6.1. tap 配置示例

以下示例配置了名为 mynet 的额外网络：

```
{
 "name": "mynet",
 "cniVersion": "0.3.1",
 "type": "tap",
 "mac": "00:11:22:33:44:55",
 "mtu": 1500,
 "selinuxcontext": "system_u:system_r:container_t:s0",
 "multiQueue": true,
 "owner": 0,
 "group": 0
 "bridge": "br1"
 }
```

24.2.3.6.2. 为TAP CNI 插件设置 SELinux 布尔值

要使用 container_t SELinux 上下文创建 tap 设备，请使用 Machine Config Operator (MCO) 在主机上启用 container_use_devices 布尔值。

先决条件

- **已安装 OpenShift CLI（oc）。**

流程

1. **创建一个新的 YAML 文件，如 setsebool-container-use-devices.yaml，详情如下：**

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: setsebool.service
        contents: |
          [Unit]
          Description=Set SELinux boolean for the TAP CNI plugin
          Before=kubelet.service

          [Service]
          Type=oneshot
          ExecStart=/usr/sbin/setsebool container_use_devices=on
          RemainAfterExit=true

          [Install]
          WantedBy=multi-user.target graphical.target
```

2. **运行以下命令来创建新的 MachineConfig 对象：**

```
$ oc apply -f setsebool-container-use-devices.yaml
```

注意

对 MachineConfig 对象应用任何更改将导致所有受影响的节点在应用更改后安全重启。此更新可能需要一些时间才能应用。

3. **运行以下命令验证是否应用了更改：**

```
$ oc get machineconfigpools
```

预期输出

```
NAME      CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master    rendered-master-e5e0c8e8be9194e7c5a882e047379cfa    True      False
False     3            3                 3                   0                   7d2h
worker    rendered-worker-d6c9ca107fba6cd76cdcbfcedcafa0f2    True      False
False     3            3                 3                   0                   7d
```

注意

所有节点都应处于更新和就绪状态。

其他资源

- 有关在节点上启用 SELinux 布尔值的更多信息，请参阅设置 SELinux 布尔值

24.2.3.7. 配置 OVN-Kubernetes 额外网络

Red Hat OpenShift Networking OVN-Kubernetes 网络插件允许为 pod 配置二级网络接口。要配置二级网络接口，您必须在 **NetworkAttachmentDefinition** 自定义资源(CR)中定义配置。

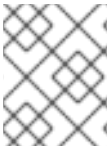
注意

Pod 和多网络策略创建可能会处于待处理状态，直到节点的 OVN-Kubernetes control plane 代理处理了关联的 **network-attachment-definition** CR。

您可以在 *第 2 层*或 *localnet* 拓扑中配置 OVN-Kubernetes 额外网络。

- 第 2 层拓扑支持 east-west 集群流量，但不允许访问底层物理网络。
- localnet 拓扑允许连接到物理网络，但需要在集群节点上进行底层 Open vSwitch (OVS) 网桥的额外配置。

以下小节提供了 OVN-Kubernetes 当前允许从属网络的每个拓扑配置示例。



注意

网络名称必须是唯一的。例如，不支持使用多个带有不同配置的 **NetworkAttachmentDefinition** CR。

24.2.3.7.1. OVN-Kubernetes 额外网络支持的平台

您可以使用以下支持的平台使用 OVN-Kubernetes 额外网络：

- 裸机
- IBM Power®
- IBM Z®
- IBM® LinuxONE

- VMware vSphere
- Red Hat OpenStack Platform(RHOSP)

24.2.3.7.2. OVN-Kubernetes 网络插件 JSON 配置表

下表描述了 OVN-Kubernetes CNI 网络插件的配置参数：

表 24.8. OVN-Kubernetes 网络插件 JSON 配置表

字段	类型	描述
cniVersion	**string**	CNI 规格版本。所需的值为 **0.3.1**。
name	**string**	网络的名称。这些网络不是命名空间。例如，您可以有一个名为 **l2-network** 的网络，该网络被在两个不同的命名空间中存在的两个不同的 **NetworkAttachmentDefinition** 引用。这样可确保，pod 可以在其自己不同的命名空间中使用的 **NetworkAttachmentDefinition** 可以通过同一二级网络进行通信。但是，这两种不同的 **NetworkAttachmentDefinition** 必须共享相同的网络特定参数，如 **topology**, **subnets**, **mtu**, 和 **excludeSubnets**。
type	**string**	用于配置的 CNI 插件的名称。这个值必须设置为 **ovn-k8s-cni-overlay**。
topology	**string**	网络的拓扑配置。必须是 **layer2** 或 **localnet** 之一。
subnets	**string**	用于集群间的网络的子网。 对于 **"topology":"layer2"** 部署，支持 IPv6 (**2001:DBB::/64**) 和双栈 (**192.168.100.0/24,2001:DBB::/64**) 子网。 在省略时，实现网络的逻辑交换机仅提供第 2 层通信，用户必须为 pod 配置 IP 地址。端口安全只阻止 MAC 欺骗。
mtu	**string**	最大传输单元 (MTU)。默认值 **1300** 由内核自动设置。
netAttachDefName	**string**	包含此配置的网络附加定义对象的元数据 **namespace** 和 **name**。例如，如果在名为 **l2-network** 的命名空间 **ns1** 中的 **NetworkAttachmentDefinition** 中定义此配置，这应设为 **ns1/l2-network**。
excludeSubnets	**string**	以逗号分隔的 CIDR 和 IP 地址列表。IP 地址从可分配的 IP 地址池中删除，永远不会传递给 pod。
vlanID	**整数**	如果拓扑设置为 **localnet**，则指定的 VLAN 标签将被分配给来自此额外网络的流量。默认为不分配 VLAN 标签。

24.2.3.7.3. 与多网络策略兼容

多网络策略 API 由 k8s.cni.cncf.io API 组中的 MultiNetworkPolicy 自定义资源定义(CRD) 提供，与 OVN-Kubernetes 二级网络兼容。在定义网络策略时，可以使用的网络策略规则取决于 OVN-Kubernetes 二级网络是否定义了 subnets 字段。详情请查看下表：

表 24.9. 支持基于 subnets CNI 配置的多网络策略选择器

指定的 subnets 字段	允许多网络策略选择器
是	• podSelector 和 namespaceSelector • ipBlock
否	• ipBlock

例如，只有在名为 blue2 的额外网络的额外网络中定义 subnets 字段时，以下多网络策略才有效：

使用 pod 选择器的多网络策略示例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: blue2
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

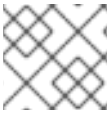
以下示例使用 ipBlock 网络策略选择器，它始终对 OVN-Kubernetes 额外网络有效：

使用 IP 块选择器的多网络策略示例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name:  ingress-ipblock
  annotations:
    k8s.v1.cni.cncf.io/policy-for: default/flatl2net
spec:
  podSelector:
    matchLabels:
      name: access-control
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.200.0.0/30
```

24.2.3.7.4. 配置第 2 层切换拓扑

交换机（层 2）拓扑网络通过集群范围的逻辑交换机互连工作负载。此配置可用于 IPv6 和双栈部署。



注意

第 2 层切换拓扑网络只允许在集群中的 pod 间传输数据数据包。

以下 JSON 示例配置交换的二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "l2-network",
  "type": "ovn-k8s-cni-overlay",
  "topology":"layer2",
  "subnets": "10.100.200.0/24",
  "mtu": 1300,
  "netAttachDefName": "ns1/l2-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

24.2.3.7.5. 配置 localnet 拓扑

交换机 (localnet) 拓扑通过集群范围的逻辑交换机将工作负载互连到物理网络。

24.2.3.7.5.1. 配置 OVN-Kubernetes 额外网络的先决条件

- 已安装 NMState Operator。如需更多信息，请参阅关于 Kubernetes NMState Operator。

24.2.3.7.5.2. 配置 OVN-Kubernetes 额外网络映射

您必须将额外网络映射到 OVN 网桥，以将其用作 OVN-Kubernetes 额外网络。网桥映射允许网络流量访问物理网络。网桥映射将物理网络名称（也称为接口标签）与通过 Open vSwitch (OVS)创建的网桥相关联。

您可以创建一个 NodeNetworkConfigurationPolicy 对象(nmstate.io/v1 API 组的一部分)，以声明性方式创建映射。此 API 由 NMState Operator 提供。通过使用此 API，您可以将网桥映射应用到与指定 nodeSelector 表达式匹配的节点，如 node-role.kubernetes.io/worker: ''。

在附加额外网络时，您可以使用现有的 br-ex 网桥或创建新网桥。使用哪种方法取决于您的特定网络基础架构。

- 如果您的节点只包含一个网络接口，则必须使用现有的网桥。这个网络接口由 OVN-Kubernetes 拥有和管理，不得从 br-ex 网桥中删除它，或更改接口配置。如果您删除或更改网络接口，您的集群网络将停止工作。
- 如果您的节点包含多个网络接口，您可以将不同的网络接口附加到新网桥，并将该网络接口用于额外网络。这种方法可用于从主集群网络进行流量隔离。

localnet1 网络在以下示例中映射到 br-ex 网桥：

共享网桥的映射示例

```
apiVersion: nmstate.io/v1
```

```
kind: NodeNetworkConfigurationPolicy
metadata:
  name: mapping 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    ovn:
      bridge-mappings:
      - localnet: localnet1 3
        bridge: br-ex 4
        state: present 5
```

1 配置对象的名称。

2 节点选择器指定要将节点网络配置策略应用到的节点。

3 流量转发到 OVS 网桥的额外网络的名称。此额外网络必须与定义 OVN-Kubernetes 额外网络的 NetworkAttachmentDefinition 对象的 spec.config.name 字段的名称匹配。

4 节点上的 OVS 网桥的名称。只有在指定 state: present 时，才需要这个值。

5 映射的状态。需要是 present（添加网桥）或 absent（删除网桥）。默认值 存在。

在以下示例中，localnet2 网络接口连接到 ovs-br1 网桥。通过此附件，网络接口可作为额外网络提供给 OVN-Kubernetes 网络插件。

具有多个接口的节点映射示例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ovs-br1-multiple-networks 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    interfaces:
    - name: ovs-br1 3
      description: |-
        A dedicated OVS bridge with eth1 as a port
        allowing all VLANs and untagged traffic
      type: ovs-bridge
      state: up
      bridge:
        options:
          stp: true
        port:
        - name: eth1 4
    ovn:
      bridge-mappings:
      - localnet: localnet2 5
        bridge: ovs-br1 6
        state: present 7
```

■

1. 配置对象的名称。
2. 节点选择器指定要将节点网络配置策略应用到的节点。
3. 新的 OVS 网桥，与 OVN-Kubernetes 用于所有集群流量的默认网桥分开。
4. 主机系统上与这个新 OVS 网桥关联的网络设备。
5. 流量转发到 OVS 网桥的额外网络的名称。此额外网络必须与定义 OVN-Kubernetes 额外网络的 NetworkAttachmentDefinition 对象的 spec.config.name 字段的名称匹配。
6. 节点上的 OVS 网桥的名称。只有在指定 state: present 时，才需要这个值。
7. 映射的状态。需要是 present（添加网桥）或 absent（删除网桥）。默认值 存在。

建议使用这个声明方法，因为 NMState Operator 将额外的网络配置应用到节点选择器指定的所有节点，并透明。

以下 JSON 示例配置 localnet 二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "ns1-localnet-network",
  "type": "ovn-k8s-cni-overlay",
  "topology":"localnet",
  "subnets": "202.10.130.112/28",
  "vlanID": 33,
  "mtu": 1500,
  "netAttachDefName": "ns1/localnet-network"
  "excludeSubnets": "10.100.200.0/29"
}
```

24.2.3.7.6. 为额外网络配置 pod

您必须通过 k8s.v1.cni.cncf.io/networks 注解来指定二级网络附加。

以下示例置备有两个二级附件的 pod，一个用于本指南中提供的每个附加配置。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: l2-network
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

24.2.3.7.7. 使用静态 IP 地址配置 pod

以下示例使用静态 IP 地址置备一个 pod。

注意

- 您只能为第 2 层附加指定 pod 的从属网络附加的 IP 地址。
- 只有在附加配置没有功能子网时，才能为 pod 指定静态 IP 地址。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "l2-network", (1)
        "mac": "02:03:04:05:06:07", (2)
        "interface": "myiface1", (3)
        "ips": [
          "192.0.2.20/24"
        ] (4)
      }
    ]'
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

1. 网络的名称。这个值在所有 NetworkAttachmentDefinition 之间必须是唯一的。
2. 为接口分配的 MAC 地址。
3. 为 pod 创建的网络接口的名称。
4. 要分配给网络接口的 IP 地址。

24.2.4. 为额外网络配置 IP 地址分配

IP 地址管理 (IPAM) Container Network Interface (CNI) 插件为其他 CNI 插件提供 IP 地址。

您可以使用以下 IP 地址分配类型：

- 静态分配。
- 通过 DHCP 服务器进行动态分配。您指定的 DHCP 服务器必须可从额外网络访问。
- 通过 Whereabouts IPAM CNI 插件进行动态分配。

24.2.4.1. 静态 IP 地址分配配置

下表描述了静态 IP 地址分配的配置：

表 24.10. ipam 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。
Routes	**数组**	指定要在 pod 中配置的路由的一组对象。
dns	**数组**	可选：指定 DNS 配置的对象数组。

address 数组需要带有以下字段的对象：

表 24.11. ipam.addresses[] array

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如：如果您指定 **10.10.21.10/24**，那么会为额外网络分配 IP 地址 **10.10.21.10**，网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 24.12. ipam.routes[] array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	网络流量路由的网关。

表 24.13. ipam.dns object

字段	类型	描述
nameservers	**数组**	用于发送 DNS 查询的一个或多个 IP 地址的数组。
domain	**数组**	要附加到主机名的默认域。例如，如果将域设置为 **example.com**，对 **example-host** 的 DNS 查找查询将被改写为 **example-host.example.com**。

字段	类型	描述
search	数组	在 DNS 查找查询过程中，附加到非限定主机名（如 **example-host**）的域名的数组。

静态 IP 地址分配配置示例

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
  }
}
```

24.2.4.2. 动态 IP 地址(DHCP)分配配置

以下 JSON 描述了使用 DHCP 进行动态 IP 地址地址分配的配置。

DHCP 租期续订

pod 在创建时获取其原始 DHCP 租期。该租期必须由集群中运行的一个小型的 DHCP 服务器部署定期续订。

要触发 DHCP 服务器的部署，您必须编辑 Cluster Network Operator 配置来创建 shim 网络附加，如下例所示：

shim 网络附加定义示例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
  # ...
```

表 24.14. ipam DHCP 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要值 dhcp。

动态 IP 地址(DHCP)分配配置示例

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

24.2.4.3. 使用 Whereabouts 进行动态 IP 地址分配配置

Whereabouts CNI 插件允许在不使用 DHCP 服务器的情况下动态地将 IP 地址分配给额外网络。

Whereabouts CNI 插件还支持在单独的 NetworkAttachmentDefinition 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

24.2.4.3.1. 动态 IP 地址配置对象

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 24.15. ipam whereabouts 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要 abouts 的值。
range	string	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	数组	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	string	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

24.2.4.3.2. 使用 Whereabouts 的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的动态地址分配配置：

Whereabouts 动态 IP 地址分配

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
```

```
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

24.2.4.3.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 2 的 **network_name** 匹配。

NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 1 的 **network_name** 匹配。

24.2.4.4. 创建abouts-reconciler 守护进程集的位置

Whereabouts 协调器负责管理集群中 pod 的动态 IP 地址分配，使用 Whereabouts IP 地址管理 (IPAM) 解决方案。它确保每个 pod 从指定的 IP 地址范围中获取唯一的 IP 地址。它还会在 pod 删除或缩减时处理 IP 地址发行版本。

注意

您还可以使用 **NetworkAttachmentDefinition** 自定义资源 (CR) 进行动态 IP 地址分配。

当您通过 Cluster Network Operator 配置一个额外的网络时，会自动创建 **whereabouts-reconciler** 守护进程集。从 YAML 清单配置额外网络时，它不会自动创建。

要触发 **whereabouts-reconciler** 守护进程集的部署，您必须通过编辑 Cluster Network Operator 自定义资源文件来手动创建一个 **whereabouts-shim** 网络附加。

使用以下步骤部署 whereabouts-reconciler 守护进程集。

流程

1. 运行以下命令来编辑 Network.operator.openshift.io 自定义资源（CR）：

```
$ oc edit network.operator.openshift.io cluster
```

2. 在自定义资源 (CR) 的 spec 定义中包含此示例 YAML extract 中显示的additionalNetworks 部分：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    rawCNIConfig: |-
      {
       "name": "whereabouts-shim",
       "cniVersion": "0.3.1",
       "type": "bridge",
       "ipam": {
         "type": "whereabouts"
       }
      }
    type: Raw
# ...
```

3. 保存文件并退出文本编辑器。

4. 运行以下命令，验证 whereabouts-reconciler 守护进程集是否已成功部署：

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

输出示例

```
pod/whereabouts-reconciler-jnp6g 1/1 Running 0 6s
pod/whereabouts-reconciler-k76gg 1/1 Running 0 6s
pod/whereabouts-reconciler-k86t9 1/1 Running 0 6s
pod/whereabouts-reconciler-p4sxw 1/1 Running 0 6s
pod/whereabouts-reconciler-rvfdv 1/1 Running 0 6s
pod/whereabouts-reconciler-svzw9 1/1 Running 0 6s
daemonset.apps/whereabouts-reconciler 6 6 6 6 6 kubernetes.io/os=linux 6s
```

24.2.4.5. 配置 Whereabouts IP 协调器调度

Whereabouts IPAM CNI 插件每天运行 IP 协调器。此过程会清理任何已搁置的 IP 分配，搁置的 IP 分配可能会耗尽 IP 资源，并使新的 pod 无法获取分配给它们的 IP。

使用这个流程更改 IP 协调器运行的频率。

先决条件

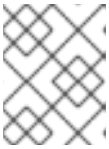
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 您已部署了whereabouts-reconciler 守护进程集，whereabouts-reconciler pod 已启动并正在运行。

流程

1. 运行以下命令，使用 IP 协调器的特定 cron 表达式在 openshift-multus 命名空间中创建一个名为 whereabouts-config 的 ConfigMap 对象：

```
$ oc create configmap whereabouts-config -n openshift-multus --from-
literal=reconciler_cron_expression="*/15 * * * *"
```

此 cron 表达式表示 IP 协调器每 15 分钟运行一次。根据您的特定要求调整表达式。



注意

whereabouts-reconciler 守护进程集只能使用包含五个星号的 cron 表达式模式。目前才不支持带有用于表示秒的第 6 个星号。

2. 运行以下命令，获取与 openshift-multus 命名空间中的 whereabouts-reconciler 守护进程集和 pod 相关的资源信息。

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

输出示例

```
pod/whereabouts-reconciler-2p7hw              1/1     Running   0          4m14s
pod/whereabouts-reconciler-76jk7              1/1     Running   0          4m14s
pod/whereabouts-reconciler-94zw6              1/1     Running   0          4m14s
pod/whereabouts-reconciler-mfh68              1/1     Running   0          4m14s
pod/whereabouts-reconciler-pgshz              1/1     Running   0          4m14s
pod/whereabouts-reconciler-xn5xz              1/1     Running   0          4m14s
daemonset.apps/whereabouts-reconciler   6         6         6       6            6
kubernetes.io/os=linux   4m16s
```

3. 运行以下命令，以验证 whereabouts-reconciler pod 是否运行带有配置的时间间隔的 IP 协调器：

```
$ oc -n openshift-multus logs whereabouts-reconciler-2p7hw
```

输出示例

```
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CHMOD
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..data_tmp":
RENAME
```

```
2024-02-02T16:33:54Z [verbose] using expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] configuration updated to file "/cron-schedule/..data". New cron expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] successfully updated CRON configuration id "00c2d1c9-631d-403f-bb86-73ad104a6817" - new cron expression: */15 * * * *
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/config": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..2024_02_02_16_26_17.3874177937": REMOVE
2024-02-02T16:45:00Z [verbose] starting reconciler run
2024-02-02T16:45:00Z [debug] NewReconcileLooper - inferred connection data
2024-02-02T16:45:00Z [debug] listing IP pools
2024-02-02T16:45:00Z [debug] no IP addresses to cleanup
2024-02-02T16:45:00Z [verbose] reconciler success
```

24.2.4.6. 为动态分配双栈 IP 地址创建配置

双栈 IP 地址分配可使用 ipRanges 参数进行配置：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 type 设置为 whereabouts。
2. 使用 ipRanges 来分配 IP 地址，如下例所示：

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
 name: cluster
spec:
 additionalNetworks:
 - name: whereabouts-shim
   namespace: default
   type: Raw
   rawCNIConfig: |-
    {
     "name": "whereabouts-dual-stack",
     "cniVersion": "0.3.1,
     "type": "bridge",
     "ipam": {
       "type": "whereabouts",
       "ipRanges": [
              {"range": "192.168.10.0/24"},
              {"range": "2001:db8::/64"}
          ]
     }
    }
```

3. 将网络附加到 pod。如需更多信息，请参阅"将 pod 添加到额外网络"。

4. 验证是否分配了所有 IP 地址。

5. 运行以下命令，以确保 IP 地址被分配为元数据。

```
$ oc exec -it mypod -- ip a
```

其他资源

- 将 pod 附加到额外网络

24.2.5. 使用 Cluster Network Operator 创建额外网络附加

Cluster Network Operator (CNO) 管理额外网络定义。当您指定要创建的额外网络时，CNO 会自动创建 NetworkAttachmentDefinition 对象。

重要

不要编辑 Cluster Network Operator 所管理的 NetworkAttachmentDefinition 对象。这样做可能会破坏额外网络上的网络流量。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 可选：为额外网络创建命名空间：

```
$ oc create namespace <namespace_name>
```

2. 要编辑 CNO 配置，请输入以下命令：

```
$ oc edit networks.operator.openshift.io cluster
```

3. 通过为您要创建的额外网络添加配置来修改您要创建的 CR，如下例所示。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks:
  - name: tertiary-net
    namespace: namespace2
    type: Raw
    rawCNIConfig: |-
      {
        "cniVersion": "0.3.1",
        "name": "tertiary-net",
        "type": "ipvlan",
```

```
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }
```

4. 保存您的更改，再退出文本编辑器以提交更改。

验证

- 通过运行以下命令确认 CNO 创建了 NetworkAttachmentDefinition 对象。CNO 创建对象之前可能会有延迟。

```
$ oc get network-attachment-definitions -n <namespace>
```

其中：

<namespace>
指定添加到 CNO 配置中的网络附加的命名空间。

输出示例

```
NAME             AGE
test-network-1   14m
```

24.2.6. 通过应用 YAML 清单来创建额外网络附加

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 使用额外网络配置创建 YAML 文件，如下例所示：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: next-net
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "work-network",
      "type": "host-device",
```

```
    "device": "eth1",
    "ipam": {
      "type": "dhcp"
    }
  }
```

2. 运行以下命令来创建额外网络：

```
$ oc apply -f <file>.yaml
```

其中：

<file>
指定包含 YAML 清单的文件名。

24.2.7. 关于在容器网络命名空间中配置 master 接口

在 OpenShift Container Platform 4.14 及更高版本中，允许用户根据容器命名空间中的 master 接口创建 MAC-VLAN、IP-VLAN 和 VLAN 子接口。

通过此功能，您可以在单独的网络附加定义中作为 pod 网络配置的一部分创建 master 接口。然后，您可以在这个接口上基于 VLAN、MACVLAN 或 IPVLAN，而无需了解节点的网络配置。

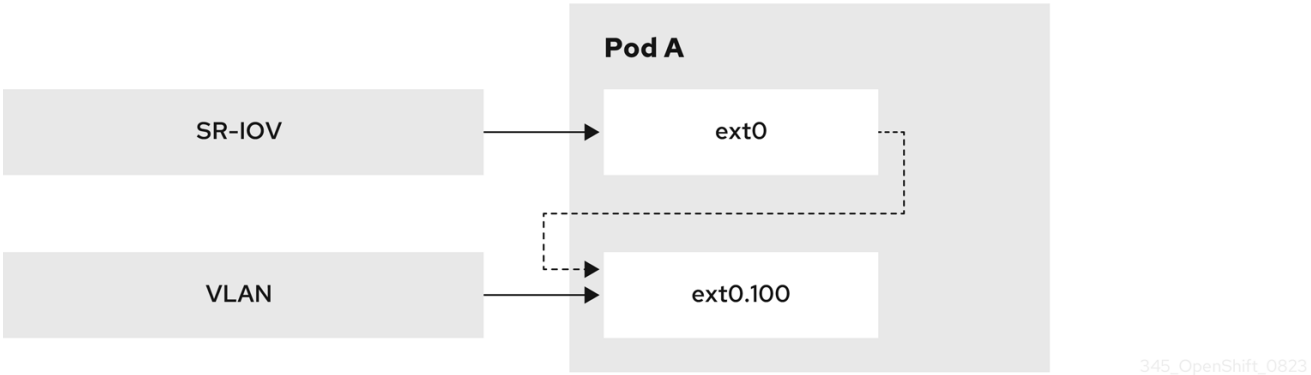
要确保容器命名空间 master 接口使用 linkInContainer，并根据额外网络的特定类型的在 VLAN、MACVLAN 或 IPVLAN 插件配置中将值设置为 true。

24.2.7.1. 在 SR-IOV VF 上创建多个 VLAN

使用此功能的一个用例是基于 SR-IOV VF 的多个 VLAN。要做到这一点，首先创建 SR-IOV 网络，然后为 VLAN 接口定义网络附加。

以下示例演示了如何配置此图中所示的设置。

图 24.1. 创建 VLAN



先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。

流程

1. 使用以下命令，创建您要部署 pod 的专用容器命名空间：

```
$ oc new-project test-namespace
```

2. 创建 SR-IOV 节点策略：

   a. 创建一个 SriovNetworkNodePolicy 对象，然后在 sriov-node-network-policy.yaml 文件中保存 YAML：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice
 isRdma: false
 needVhostNet: true
 nicSelector:
  vendor: "15b3" 1
  deviceID: "101b" 2
  rootDevices: ["00:05.0"]
 numVfs: 10
 priority: 99
 resourceName: sriovnic
 nodeSelector:
   feature.node.kubernetes.io/network-sriov.capable: "true"
```

注意

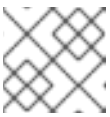
SR-IOV 网络节点策略配置示例，使用设置 deviceType: netdevice，专为 Mellanox 网络接口卡(NIC)量身定制。

1 SR-IOV 网络设备厂商的十六进制代码。值 15b3 代表与 Mellanox NIC 关联。

2 SR-IOV 网络设备的设备十六进制代码。

   b. 运行以下命令来应用 YAML：

```
$ oc apply -f sriov-node-network-policy.yaml
```



注意

应用这可能需要一些时间，因为需要重新引导的节点。

3. 创建 SR-IOV 网络：

   a. 为额外 SR-IOV 网络附加创建 SriovNetwork 自定义资源(CR)，如下例所示。将 YAML 保存为文件 sriov-network-attachment.yaml ：

```
apiVersion: sriovnetwork.openshift.io/v1
```

```
kind: SriovNetwork
metadata:
 name: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
 resourceName: sriovnic
 spoofChk: "off"
 trust: "on"
```

b. 运行以下命令来应用 YAML：

```
$ oc apply -f sriov-network-attachment.yaml
```

4. 创建 VLAN 额外网络：

a. 使用以下 YAML 示例，创建一个名为 ipvlan100-additional-network-configuration.yaml 的文件：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: vlan-100
  namespace: test-namespace
spec:
  config: |
    {
      "cniVersion": "0.4.0",
      "name": "vlan-100",
      "plugins": [
        {
          "type": "vlan",
          "master": "ext0", 1
          "mtu": 1500,
          "vlanId": 100,
          "linkInContainer": true, 2
          "ipam": {"type": "whereabouts", "ipRanges": [{"range": "1.1.1.0/24"}]}
        }
      ]
    }
```

1 VLAN 配置需要指定 master 名称。这可以在 pod 网络注解中配置。

2 必须指定 linkInContainer 参数。

b. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f vlan100-additional-network-configuration.yaml
```

5. 使用之前指定的网络创建 pod 定义：

a. 使用以下 YAML 示例，创建一个名为 pod-a.yaml 文件的文件：

注意

以下清单包括 2 个资源：

- 带有安全标签的命名空间
- 带有适当的网络注解的 Pod 定义

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "sriov-network",
        "namespace": "test-namespace",
        "interface": "ext0" 1
      },
      {
        "name": "vlan-100",
        "namespace": "test-namespace",
        "interface": "ext0.100"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
  containers:
    - name: nginx-container
      image: nginxinc/nginx-unprivileged:latest
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      ports:
        - containerPort: 80
      seccompProfile:
        type: "RuntimeDefault"
```

1 用作 VLAN 接口主接口的名称。

b. 运行以下命令来应用 YAML 文件：

-

```
$ oc apply -f pod-a.yaml
```

6. 运行以下命令，在 test-namespace 中获取 nginx-pod 的详细信息：

```
$ oc describe pods nginx-pod -n test-namespace
```

输出示例

```
Name:         nginx-pod
Namespace:    test-namespace
Priority:     0
Node:         worker-1/10.46.186.105
Start Time:   Mon, 14 Aug 2023 16:23:13 -0400
Labels:       <none>
Annotations:  k8s.ovn.org/pod-networks:
                {"default":{"ip_addresses":
["10.131.0.26/23"],"mac_address":"0a:58:0a:83:00:1a","gateway_ips":
["10.131.0.1"],"routes":[{"dest":"10.128.0.0...
              k8s.v1.cni.cncf.io/network-status:
               [{
                  "name": "ovn-kubernetes",
                  "interface": "eth0",
                  "ips": [
                     "10.131.0.26"
                  ],
                  "mac": "0a:58:0a:83:00:1a",
                  "default": true,
                  "dns": {}
               },{
                  "name": "test-namespace/sriov-network",
                  "interface": "ext0",
                  "mac": "6e:a7:5e:3f:49:1b",
                  "dns": {},
                  "device-info": {
                     "type": "pci",
                     "version": "1.0.0",
                     "pci": {
                        "pci-address": "0000:d8:00.2"
                     }
                  }
               },{
                  "name": "test-namespace/vlan-100",
                  "interface": "ext0.100",
                  "ips": [
                     "1.1.1.1"
                  ],
                  "mac": "6e:a7:5e:3f:49:1b",
                  "dns": {}
               }]
              k8s.v1.cni.cncf.io/networks:
               [ { "name": "sriov-network", "namespace": "test-namespace", "interface":
"ext0" }, { "name": "vlan-100", "namespace": "test-namespace", "i...
              openshift.io/scc: privileged
Status:       Running
```

```
IP:          10.131.0.26
IPs:
  IP:  10.131.0.26
```

24.2.7.2. 基于容器命名空间中的网桥主接口创建子接口

创建子接口可应用到其他类型的接口。按照以下步骤，基于容器命名空间中的网桥 master 接口创建子接口。

先决条件

- 已安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 运行以下命令，创建您要部署 pod 的专用容器命名空间：

```
$ oc new-project test-namespace
```

2. 使用以下 YAML 示例，创建一个名为 bridge-nad.yaml 的桥接 NetworkAttachmentDefinition 自定义资源 (CR) 文件：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "bridge-network",
    "type": "bridge",
    "bridge": "br-001",
    "isGateway": true,
    "ipMasq": true,
    "hairpinMode": true,
    "ipam": {
      "type": "host-local",
      "subnet": "10.0.0.0/24",
      "routes": [{"dst": "0.0.0.0/0"}]
    }
  }'
```

3. 运行以下命令，将 NetworkAttachmentDefinition CR 应用到 OpenShift Container Platform 集群：

```
$ oc apply -f bridge-nad.yaml
```

4. 运行以下命令验证 NetworkAttachmentDefinition CR 是否已成功创建：

```
$ oc get network-attachment-definitions
```

输出示例

```
NAME            AGE
bridge-network   15s
```

5. 使用以下 YAML 示例，为 IPVLAN 额外网络配置创建一个名为 ipvlan-additional-network-configuration.yaml 的文件：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ipvlan-net
  namespace: test-namespace
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "ipvlan-net",
    "type": "ipvlan",
    "master": "ext0", 1
    "mode": "l3",
    "linkInContainer": true, 2
    "ipam": {"type": "whereabouts", "ipRanges": [{"range": "10.0.0.0/24"}]}
  }'
```

1. 指定要与网络附加关联的以太网接口。这会随后在 pod 网络注解中配置。
2. 指定 master 接口位于容器网络命名空间中。

6. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f ipvlan-additional-network-configuration.yaml
```

7. 运行以下命令验证 NetworkAttachmentDefinition CR 是否已成功创建：

```
$ oc get network-attachment-definitions
```

输出示例

```
NAME            AGE
bridge-network   87s
ipvlan-net       9s
```

8. 使用以下 YAML 示例，为 pod 定义创建一个名为 pod-a.yaml 的文件：

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
```

```
      "name": "bridge-network",
      "interface": "ext0" 1
    },
    {
      "name": "ipvlan-net",
      "interface": "ext1"
    }
  ]'
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-pod
    image: quay.io/openshifttest/hello-
sdn@sha256:c89445416459e7adea9a5a416b3365ed3d74f2491beb904d61dc8d1eb89a72
a4
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1 指定用作 IPVLAN 接口的 master 的名称。

9. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f pod-a.yaml
```

10. 使用以下命令验证 pod 是否正在运行：

```
$ oc get pod -n test-namespace
```

输出示例

```
NAME   READY   STATUS    RESTARTS   AGE
pod-a  1/1     Running   0          2m36s
```

11. 运行以下命令，显示 test-namespace 中 pod-a 资源的网络接口信息：

```
$ oc exec -n test-namespace pod-a -- ip a
```

输出示例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if105: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue
state UP group default
```

```
    link/ether 0a:58:0a:d9:00:5d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.217.0.93/23 brd 10.217.1.255 scope global eth0
      valid_lft forever preferred_lft forever
    inet6 fe80::488b:91ff:fe84:a94b/64 scope link
      valid_lft forever preferred_lft forever
4: ext0@if107: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.2/24 brd 10.0.0.255 scope global ext0
      valid_lft forever preferred_lft forever
    inet6 fe80::bcda:bdff:fe7e:f437/64 scope link
      valid_lft forever preferred_lft forever
5: ext1@ext0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global ext1
      valid_lft forever preferred_lft forever
    inet6 fe80::beda:bd00:17e:f437/64 scope link
      valid_lft forever preferred_lft forever
```

此输出显示，网络接口 ext1 与物理接口 ext0 关联。

24.3. 关于虚拟路由和转发

24.3.1. 关于虚拟路由和转发

虚拟路由和转发（VRF）设备与 IP 规则相结合，提供了创建虚拟路由和转发域的能力。VRF 减少了 CNF 所需的权限数量，并可提高二级网络网络拓扑的可见性。VRF 用于提供多租户功能，例如，每个租户都有自己的唯一的路由表且需要不同的默认网关。

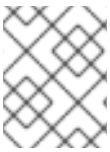
进程可将套接字绑定到 VRF 设备。通过绑定套接字的数据包使用与 VRF 设备关联的路由表。VRF 的一个重要特性是，它只影响 OSI 模型层 3 以上的流量，因此 L2 工具（如 LLDP）不会受到影响。这可让优先级更高的 IP 规则（如基于策略的路由）优先于针对特定流量的 VRF 设备规则。

24.3.1.1. 这对针对电信业使用的 pod 的从属网络提供了好处

在电信业，每个 CNF 都可连接到共享相同地址空间的多个不同的网络。这些从属网络可能会与集群的主网络 CIDR 冲突。使用 CNI VRF 插件，网络功能可使用相同的 IP 地址连接到不同的客户基础架构，使不同的客户保持隔离。IP 地址与 OpenShift Container Platform IP 空间重叠。CNI VRF 插件还可减少 CNF 所需的权限数量，并提高从属网络的网络拓扑的可见性。

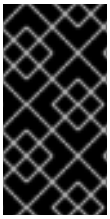
24.4. 配置多网络策略

作为集群管理员，您可以为单根 I/O 虚拟化 (SR-IOV)、MAC Virtual Local Area Network (MacVLAN) 或 OVN-Kubernetes 额外网络配置多网络策略。macvlan 额外网络被完全支持。不支持其他类型的额外网络，如 IP Virtual Local Area Network (IPVLAN)。



注意

支持为 SR-IOV 额外网络配置多网络策略，仅支持内核网络接口控制器 (NIC)。SR-IOV 不支持 Data Plane Development Kit (DPDK)应用程序。



重要

支持为 SR-IOV 额外网络配置多网络策略是技术预览功能，且只支持内核网络接口卡 (NIC)。SR-IOV 不支持 Data Plane Development Kit (DPDK)应用程序。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

24.4.1. 多网络策略和网络策略之间的区别

虽然 MultiNetworkPolicy API 实现NetworkPolicy API，但有几个重要的区别：

- 您必须使用 MultiNetworkPolicy API：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
```

- 当使用 CLI 与多网络策略交互时，您必须使用 multi-networkpolicy 资源名称。例如，您可以使用 oc get multi-networkpolicy <name> 命令来查看多网络策略对象，其中 <name> 是多网络策略的名称。

- 您必须使用定义 macvlan 或 SR-IOV 额外网络的网络附加定义名称指定一个注解：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
```

其中：

<network_name>
指定网络附加定义的名称。

24.4.2. 为集群启用多网络策略

作为集群管理员，您可以在集群中启用多网络策略支持。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 使用以下 YAML 创建 multinetwork-enable-patch.yaml 文件：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  useMultiNetworkPolicy: true
```

2. 配置集群以启用多网络策略：

```
$ oc patch network.operator.openshift.io cluster --type=merge --patch-
file=multinetwork-enable-patch.yaml
```

输出示例

```
network.operator.openshift.io/cluster patched
```

24.4.3. 在 IPv6 网络中支持多网络策略

ICMPv6 Neighbor Discovery Protocol (NDP) 是一组消息和流程，使设备能够发现和维护有关邻居节点的信息。NDP 在 IPv6 网络中扮演着关键角色，用于处理同一链路上的设备间的交互。

当 **useMultiNetworkPolicy** 参数被设置为 **true** 时，Cluster Network Operator (CNO) 会部署多网络策略的 iptables 实现。

要在 IPv6 网络中支持多网络策略，Cluster Network Operator 会在受多网络策略影响的每个 pod 中部署以下规则集：

多网络策略自定义规则

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: multi-networkpolicy-custom-rules
  namespace: openshift-multus
data:

  custom-v6-rules.txt: |
    # accept NDP
    -p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT 1
    -p icmpv6 --icmpv6-type neighbor-advertisement -j ACCEPT 2
    # accept RA/RS
    -p icmpv6 --icmpv6-type router-solicitation -j ACCEPT 3
    -p icmpv6 --icmpv6-type router-advertisement -j ACCEPT 4
```

1. 此规则允许传入的 ICMPv6 邻居请求消息，它们是邻居发现协议 (NDP) 的一部分。这些消息帮助确定邻居节点的链路层地址。
2. 此规则允许传入 ICMPv6 邻居公告消息，它们是 NDP 的一部分，并提供有关发件人链路层地址的信息。
3. 此规则允许传入的 ICMPv6 路由器请求消息。主机使用这些消息来请求路由器配置信息。
4. 此规则允许传入的 ICMPv6 路由器广告消息，为主机提供配置信息。

注意

您不能编辑这些预定义规则。

这些规则共同启用基本的 ICMPv6 流量，以便正确实现网络功能，包括 IPv6 环境中的地址解析和路由器通信。使用这些规则以及多网络策略拒绝流量时，应用程序应不会遇到连接问题。

24.4.4. 使用多网络策略

作为集群管理员，您可以创建、编辑、查看和删除多网络策略。

24.4.4.1. 先决条件

- 您已为集群启用了多网络策略支持。

24.4.4.2. 使用 CLI 创建多网络策略

要定义细致的规则来描述集群中命名空间允许的入口或出口网络流量，您可以创建一个多网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

1. 创建策略规则：
   a. 创建一个 <policy_name>.yaml 文件：

   ```
   $ touch <policy_name>.yaml
   ```

   其中：

   <policy_name>
   指定多网络策略文件名。

   b. 在您刚才创建的文件中定义多网络策略，如下例所示：

   拒绝来自所有命名空间中的所有 pod 的入口流量

   这是一个基本的策略，阻止配置其他网络策略所允许的跨 pod 流量以外的所有跨 pod 网络。

   ```
   apiVersion: k8s.cni.cncf.io/v1beta1
   kind: MultiNetworkPolicy
   metadata:
     name: deny-by-default
     annotations:
       k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
   spec:
     podSelector: {}
   ```

```
  policyTypes:
  - Ingress
  ingress: []
```

其中：

<network_name>
指定网络附加定义的名称。

允许来自所有命名空间中的所有 pod 的入口流量

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

其中：

<network_name>
指定网络附加定义的名称。

允许从特定命名空间中到一个 pod 的入口流量

此策略允许流量从在 namespace-y 中运行的容器集到标记 pod-a 的 pod。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-traffic-pod
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
   matchLabels:
      pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

其中：

<network_name>
指定网络附加定义的名称。

限制到服务的流量

应用此策略可确保每个带有标签 app=bookstore 和标签 role=api 的 pod 只能被带有标签 app=bookstore 的 pod 访问。在本例中，应用可以是 REST API 服务器，标记为标签 app=bookstore 和 role=api。

这个示例可以解决了以下用例：

- 将到一个服务的流量限制为仅使用需要它的其他微服务。
- 将连接限制为只允许使用它的应用程序。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: api-allow
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

其中：

<network_name>
指定网络附加定义的名称。

2. 运行以下命令来创建多网络策略对象：

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

其中：

<policy_name>
指定多网络策略文件名。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```

注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式创建网络策略。

24.4.4.3. 编辑多网络策略

您可以编辑命名空间中的多网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在存在多网络策略的命名空间中工作。

流程

1. 可选：要列出命名空间中的多网络策略对象，请输入以下命令：

```
$ oc get multi-networkpolicy
```

其中：

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

2. 编辑多网络策略对象。
   - 如果您在文件中保存了多网络策略定义，请编辑该文件并进行必要的更改，然后输入以下命令。

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

其中：

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

<policy_file>
指定包含网络策略的文件的名称。

   - 如果您需要直接更新多网络策略对象，请输入以下命令：

```
$ oc edit multi-networkpolicy <policy_name> -n <namespace>
```

其中：

<policy_name>
指定网络策略的名称。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

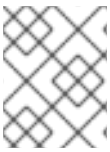
3. 确认已更新多网络策略对象。

```
$ oc describe multi-networkpolicy <policy_name> -n <namespace>
```

其中：

<policy_name>
指定多网络策略的名称。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。



注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或通过 Actions 菜单从 web 控制台中的策略编辑网络策略。

24.4.4.4. 使用 CLI 查看多网络策略

您可以检查命名空间中的多网络策略。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在存在多网络策略的命名空间中工作。

流程

- 列出命名空间中的多网络策略：
  - 要查看命名空间中定义的多网络策略对象，请输入以下命令：

    ```
    $ oc get multi-networkpolicy
    ```

  - 可选：要检查特定的多网络策略，请输入以下命令：

    ```
    $ oc describe multi-networkpolicy <policy_name> -n <namespace>
    ```

    其中：

    <policy_name>
    指定要检查的多网络策略的名称。

    <namespace>
    可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

注意

如果您使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式查看网络策略。

24.4.4.5. 使用 CLI 删除多网络策略

您可以删除命名空间中的多网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在存在多网络策略的命名空间中工作。

流程

- 要删除多网络策略对象，请输入以下命令：

```
$ oc delete multi-networkpolicy <policy_name> -n <namespace>
```

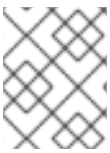
其中：

<policy_name>
指定多网络策略的名称。

<namespace>
可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
```



注意

如果使用 cluster-admin 权限登录到 web 控制台，您可以选择在集群上以 YAML 或通过 Actions 菜单从 web 控制台中的策略删除网络策略。

24.4.4.6. 创建默认拒绝所有多网络策略

这是一个基本的策略，阻止其他部署网络策略允许的网络流量以外的所有跨 pod 网络。此流程强制使用默认 deny-by-default 策略。

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。

- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

1. 创建以下 YAML，以定义 deny-by-default 策略，以拒绝所有命名空间中的所有 pod 的入口流量。将 YAML 保存到 deny-by-default.yaml 文件中：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  namespace: default 1
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <namespace_name>/<network_name> 2
spec:
  podSelector: {} 3
  policyTypes: 4
  - Ingress 5
  ingress: [] 6
```

1. namespace：default 将此策略部署到 default 命名空间。
2. network_name ：指定一个网络附加定义的名称。
3. podSelector: 为空，这意味着它与所有 pod 匹配。因此，该策略适用于 default 命名空间中的所有 pod。
4. policyTypes ： NetworkPolicy 相关的规则类型列表。
5. 指定为 Ingress only policyType。
6. 没有指定 ingress 规则。这会导致传入的流量丢弃至所有 pod。

2. 输入以下命令应用策略：

```
$ oc apply -f deny-by-default.yaml
```

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```

24.4.4.7. 创建多网络策略以允许来自外部客户端的流量

使用 deny-by-default 策略，您可以继续配置策略，允许从外部客户端到带有标签 app=web 的 pod 的流量。

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置策略，以直接从公共互联网允许外部服务，或使用 Load Balancer 访问 pod。只有具有标签 app=web 的 pod 才允许流量。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

1. 创建策略，以直接从公共互联网的流量或使用负载均衡器访问 pod。将 YAML 保存到 web-allow-external.yaml 文件中：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-external
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

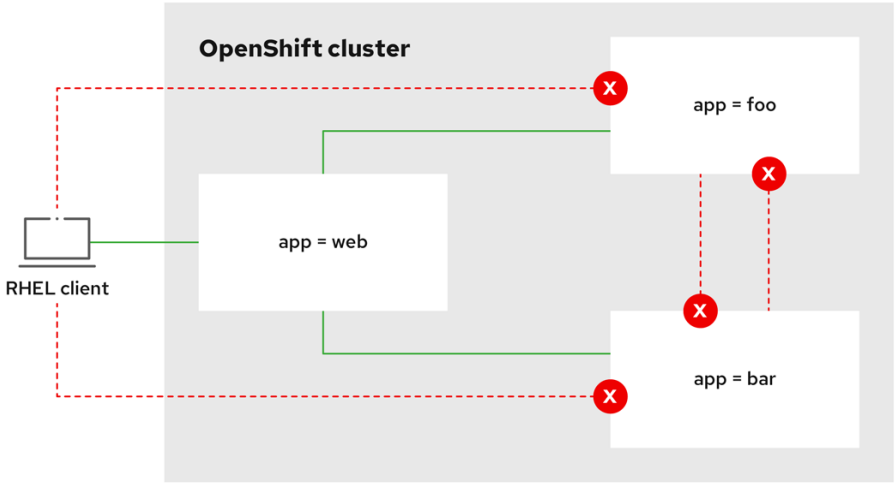
2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-external.yaml
```

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-external created
```

此策略允许来自所有资源的流量，包括下图所示的外部流量：



24.4.4.8. 创建一个多网络策略，允许从所有命名空间中到应用程序的流量

注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置允许从所有命名空间中的所有 pod 流量到特定应用程序的策略。

先决条件

- **集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。**
- **已安装 OpenShift CLI（oc）。**
- **使用具有 cluster-admin 权限的用户登陆到集群。**
- **您在多网络策略应用到的命名空间中工作。**

流程

1. **创建一个策略，允许从所有命名空间中的所有 pod 流量到特定应用。将 YAML 保存到 web-allow-all-namespaces.yaml 文件中：**

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-all-namespaces
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
  - Ingress
```

```
  ingress:
  - from:
    - namespaceSelector: {} 2
```

1 仅将策略应用到 default 命名空间中的 app:web pod。

2 选择所有命名空间中的所有 pod。

注意

默认情况下，如果您省略了指定 namespaceSelector 而不是选择任何命名空间，这意味着策略只允许从网络策略部署到的命名空间的流量。

2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-all-namespaces.yaml
```

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-all-namespaces created
```

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --
port=80
```

2. 运行以下命令在 secondary 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. 在 shell 中运行以下命令，并观察是否允许请求：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

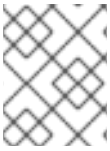
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

24.4.4.9. 创建多网络策略允许从命名空间中到应用程序的流量



注意

如果使用具有 cluster-admin 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

按照以下步骤配置允许从特定命名空间中到带有 app=web 标签的 pod 的策略。您可能需要进行以下操作：

- 将流量限制为部署生产工作负载的命名空间。
- 启用部署到特定命名空间的监控工具，以从当前命名空间中提取指标。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

1. 创建一个策略，允许来自特定命名空间中所有 pod 的流量，其标签为 purpose=production。将 YAML 保存到 web-allow-prod.yaml 文件中：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-prod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
```

```
  - Ingress
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        purpose: production 2
```

1 仅将策略应用到 default 命名空间中的 app:web pod。

2 将流量仅限制为具有标签 purpose=production 的命名空间中的 pod。

2. 输入以下命令应用策略：

```
$ oc apply -f web-allow-prod.yaml
```

输出示例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-prod created
```

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --
port=80
```

2. 运行以下命令来创建 prod 命名空间：

```
$ oc create namespace prod
```

3. 运行以下命令来标记 prod 命名空间：

```
$ oc label namespace/prod purpose=production
```

4. 运行以下命令来创建 dev 命名空间：

```
$ oc create namespace dev
```

5. 运行以下命令来标记 dev 命名空间：

```
$ oc label namespace/dev purpose=testing
```

6. 运行以下命令在 dev 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```

7. 在 shell 中运行以下命令，并观察请求是否被阻止：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

```
wget: download timed out
```

8. 运行以下命令，在 prod 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. 在 shell 中运行以下命令，并观察是否允许请求：

```
# wget -qO- --timeout=2 http://web.default
```

预期输出

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

24.4.5. 其他资源

- 关于网络策略
- 了解多网络
- 配置 macvlan 网络
- 配置 SR-IOV 网络设备

24.5. 将 POD 附加到额外网络

作为集群用户，您可以将 pod 附加到额外网络。

24.5.1. 将 pod 添加到额外网络

您可以将 pod 添加到额外网络。pod 继续通过默认网络发送与集群相关的普通网络流量。

创建 pod 时会附加额外网络。但是，如果 pod 已存在，您无法为其附加额外网络。

pod 必须与额外网络处于相同的命名空间。

先决条件

- 安装 OpenShift CLI（oc）。
- 登录到集群。

流程

1. 为 Pod 对象添加注解。只能使用以下注解格式之一：
   a. 要在没有自定义的情况下附加额外网络，请使用以下格式添加注解。将 <network> 替换为要与 pod 关联的额外网络的名称：

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

1 要指定多个额外网络，请使用逗号分隔各个网络。逗号之间不可包括空格。如果您多次指定同一额外网络，则该 pod 会将多个网络接口附加到该网络。

   b. 要通过自定义来附加额外网络，请添加具有以下格式的注解：

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
```

1 指定 NetworkAttachmentDefinition 对象定义的额外网络的名称。

2 指定定义 NetworkAttachmentDefinition 对象的命名空间。

3 可选：为默认路由指定覆盖，如 192.168.17.1。

2. 运行以下命令来创建 pod。将 <name> 替换为 pod 的名称。

```
$ oc create -f <name>.yaml
```

3. 可选：要确认 Pod CR 中是否存在注解，请输入以下命令将<name> 替换为 pod 的名称。

```
$ oc get pod <name> -o yaml
```

在以下示例中，example-pod pod 附加到 net1 额外网络：

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1 k8s.v1.cni.cncf.io/network-status 参数是对象的 JSON 数组。每个对象描述附加到 pod 的额外网络的状态。注解值保存为纯文本值。

24.5.1.1. 指定特定于 pod 的地址和路由选项

将 pod 附加到额外网络时，您可能需要在特定 pod 中指定有关该网络的其他属性。这可让您更改路由的某些方面，并指定静态 IP 地址和 MAC 地址。要达到此目的，您可以使用 JSON 格式的注解。

先决条件

- pod 必须与额外网络处于相同的命名空间。
- 安装 OpenShift CLI (oc) 。
- 您必须登录集群。

流程

要在指定地址和/或路由选项的同时将 pod 添加到额外网络，请完成以下步骤：

1. 编辑 Pod 资源定义。如果要编辑现有 Pod 资源，请运行以下命令在默认编辑器中编辑其定义。将 <name> 替换为要编辑的 Pod 资源的名称。

```
$ oc edit pod <name>
```

2. 在 Pod 资源定义中，将 k8s.v1.cni.cncf.io/networks 参数添加到 pod metadata 映射中。k8s.v1.cni.cncf.io/networks 接受 JSON 字符串，该字符串除指定附加属性外，还引用 NetworkAttachmentDefinition 自定义资源（CR）名称的对象。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]' 1
```

1 将 <network> 替换为 JSON 对象，如下例所示。单引号是必需的。

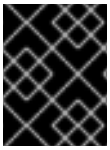
3. 在以下示例中，通过 default-route 参数，注解指定了哪个网络附加将使用默认路由。

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "net1"
    },
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }]'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

1 name 是与 pod 关联的额外网络的名称。

2 default-route 指定了一个网关，当在路由表中没有其它路由条目时使用这个网关。如果指定了多个 default-route 键，这将导致 pod 无法成为活跃状态。

默认路由将导致任何没有在其它路由中指定的流量被路由到网关。

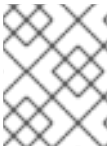


重要

将 OpenShift Container Platform 的默认路由设置为默认网络接口以外的接口时，可能会导致应该是 pod 和 pod 间的网络流量被路由到其他接口。

要验证 pod 的路由属性，可使用 oc 命令在 pod 中执行 ip 命令。

```
$ oc exec -it <pod_name> -- ip route
```



注意

您还可以引用 pod 的 k8s.v1.cni.cncf.io/network-status 来查看哪个额外网络已被分配默认路由，这可以通过 JSON 格式的对象列表中的 default-route 键存在。

要为 pod 设置静态 IP 地址或 MAC 地址，您可以使用 JSON 格式的注解。这要求您创建允许此功能的网络。这可以在 CNO 的 rawCNIConfig 中指定。

1. 运行以下命令来编辑 CNO CR：

```
$ oc edit networks.operator.openshift.io cluster
```

以下 YAML 描述了 CNO 的配置参数：

Cluster Network Operator YAML 配置

```
name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
type: Raw
```

1. 为您要创建的额外网络附加指定名称。该名称在指定的 namespace 中需要是唯一的。
2. 指定要在其中创建网络附加的命名空间。如果您未指定值，则使用 default 命名空间。
3. 基于以下模板，以 JSON 格式指定 CNI 插件配置。

以下对象描述了使用 macvlan CNI 插件的静态 MAC 地址和 IP 地址的配置参数：

使用静态 IP 和 MAC 地址的 macvlan CNI 插件 JSON 配置对象

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "plugins": [{ 2
     "type": "macvlan",
     "capabilities": { "ips": true }, 3
     "master": "eth0", 4
     "mode": "bridge",
     "ipam": {
       "type": "static"
     }
    }, {
     "capabilities": { "mac": true }, 5
     "type": "tuning"
    }]
}
```

1. 指定要创建的额外网络附加的名称。该名称在指定的 namespace 中需要是唯一的。

2 指定 CNI 插件配置的数组。第一个对象指定 macvlan 插件配置，第二个对象指定 tuning 插件配置。

3 指定一个请求启用 CNI 插件运行时配置功能的静态 IP 地址功能。

4 指定 macvlan 插件使用的接口。

5 指定一个请求启用 CNI 插件的静态 MAC 地址功能。

以上网络附加可能会以 JSON 格式的注解引用，同时使用相关的键来指定将哪些静态 IP 和 MAC 地址分配给指定 pod。

使用以下内容编辑 pod：

```
$ oc edit pod <name>
```

使用静态 IP 和 MAC 地址的 macvlan CNI 插件 JSON 配置对象

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "<name>", 1
        "ips": [ "192.0.2.205/24" ], 2
        "mac": "CA:FE:C0:FF:EE:00" 3
      }
    ]'
```

1 使用在创建 rawCNIConfig 时提供的 <name> 。

2 提供包括子网掩码的 IP 地址。

3 提供 MAC 地址。

注意

静态 IP 地址和 MAC 地址不需要同时使用，您可以单独使用，也可以一起使用。

要验证一个带有额外网络的 pod 的 IP 地址和 MAC 属性，请使用 oc 命令在 pod 中执行 ip 命令。

```
$ oc exec -it <pod_name> -- ip a
```

24.6. 从额外网络中删除 POD

作为集群用户，您可以从额外网络中删除 pod。

24.6.1. 从额外网络中删除 pod

您只能通过删除 pod 来从额外网络中删除 pod。

先决条件

- 一个额外网络被附加到 pod。
- 安装 OpenShift CLI（oc）。
- 登录到集群。

流程

- 要删除 pod，输入以下命令：

```
$ oc delete pod <name> -n <namespace>
```

  - <name> 是 pod 的名称。
  - <namespace> 是包含 pod 的命名空间。

24.7. 编辑额外网络

作为集群管理员，您可以修改现有额外网络的配置。

24.7.1. 修改额外网络附加定义

作为集群管理员，您可以对现有额外网络进行更改。任何附加到额外网络的现有 pod 都不会被更新。

先决条件

- 已为集群配置了额外网络。
- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

要为集群编辑额外网络，请完成以下步骤：

1. 运行以下命令，在默认文本编辑器中编辑 Cluster Network Operator (CNO) CR：

```
$ oc edit networks.operator.openshift.io cluster
```

2. 在 additionalNetworks 集合中，用您的更改更新额外网络。
3. 保存您的更改，再退出文本编辑器以提交更改。
4. 可选：运行以下命令确认 CNO 更新了 NetworkAttachmentDefinition 对象。将 <network-name> 替换为要显示的额外网络名称。CNO 根据您的更改更新 NetworkAttachmentDefinition 对象前可能会有延迟。

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

   例如，以下控制台输出显示名为 net1 的 NetworkAttachmentDefinition 对象：

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n"
```

```
.spec.config}}'
{ "cniVersion": "0.3.1", "type": "macvlan",
"master": "ens5",
"mode": "bridge",
"ipam":      {"type":"static","routes":
[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}],"dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}} }
```

24.8. 删除额外网络

作为集群管理员，您可以删除额外网络附加。

24.8.1. 删除额外网络附加定义

作为集群管理员，您可以从 OpenShift Container Platform 集群中删除额外网络。额外网络不会从它所附加的任何 pod 中删除。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

要从集群中删除额外网络，请完成以下步骤：

1. 运行以下命令，在默认文本编辑器中编辑 Cluster Network Operator (CNO)：

```
$ oc edit networks.operator.openshift.io cluster
```

2. 从您要删除的网络附加定义的 additionalNetworks 集合中删除配置，以此修改 CR。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] 1
```

1 如果要删除 additionalNetworks 集合中唯一额外网络附加定义的配置映射，您必须指定一个空集合。

3. 保存您的更改，再退出文本编辑器以提交更改。

4. 可选：通过运行以下命令确认删除了额外网络 CR：

```
$ oc get network-attachment-definition --all-namespaces
```

24.9. 为 VRF 分配从属网络

作为集群管理员，您可以使用 CNI VRF 插件为虚拟路由和转发 (VRF) 域配置额外网络。此插件创建的虚拟网络与您指定的物理接口关联。

使用带有 VRF 实例的二级网络有以下优点：

工作负载隔离

通过为额外网络配置 VRF 实例来隔离工作负载流量。

提高了安全性

通过 VRF 域中的隔离网络路径启用更高的安全性。

多租户支持

通过网络分段支持每个租户的 VRF 域中唯一路由表的多租户。

注意

使用 VRF 的应用程序必须绑定到特定设备。通常的用法是在套接字中使用 SO_BINDTODEVICE 选项。SO_BINDTODEVICE 选项将套接字绑定到在传递接口名称中指定的设备，如 eth1。要使用 SO_BINDTODEVICE 选项，应用程序必须具有 CAP_NET_RAW 功能。

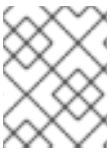
OpenShift Container Platform pod 不支持通过 ip vrf exec 命令使用 VRF。要使用 VRF，将应用程序直接绑定到 VRF 接口。

其他资源

- 关于虚拟路由和转发

24.9.1. 使用 CNI VRF 插件创建额外网络附加

Cluster Network Operator (CNO) 管理额外网络定义。当您指定要创建的额外网络时，CNO 会自动创建 NetworkAttachmentDefinition 自定义资源（CR）。



注意

请勿编辑 Cluster Network Operator 所管理的 NetworkAttachmentDefinition CR。这样做可能会破坏额外网络上的网络流量。

要使用 CNI VRF 插件创建额外网络附加，请执行以下步骤。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift 集群。

流程

1. 为额外网络附加创建 Network 自定义资源 (CR)，并为额外网络插入 rawCNIConfig 配置，如下例所示。将 YAML 保存为文件 additional-network-attachment.yaml。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
```

```
  name: cluster
spec:
  additionalNetworks:
   - name: test-network-1
     namespace: additional-network-1
     type: Raw
     rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "macvlan-vrf",
      "plugins": [ 1
      {
        "type": "macvlan",
        "master": "eth1",
        "ipam": {
           "type": "static",
           "addresses": [
           {
              "address": "191.168.1.23/24"
           }
           ]
        }
      },
      {
        "type": "vrf", 2
        "vrfname": "vrf-1", 3
        "table": 1001 4
      }]
     }'
```

1 插件必须是一个列表。列表中的第一个项必须是支持 VRF 网络的从属网络。列表中的第二个项目是 VRF 插件配置。

2 type 必须设为 vrf。

3 vrfname 是接口分配的 VRF 的名称。如果 pod 中不存在，则创建它。

4 可选。table 是路由表 ID。默认情况下使用 tableid 参数。如果没有指定，CNI 会为 VRF 分配免费路由表 ID。

注意

只有在资源类型为 netdevice 时，VRF 才能正常工作。

2. 创建 Network 资源：

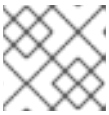
```
$ oc create -f additional-network-attachment.yaml
```

3. 通过运行以下命令确认 CNO 创建了 NetworkAttachmentDefinition CR。将 <namespace> 替换为您在配置网络附加时指定的命名空间，如 additional-network-1。

```
$ oc get network-attachment-definitions -n <namespace>
```

输出示例

```
NAME                    AGE
additional-network-1    14m
```



注意

CNO 创建 CR 之前可能会有延迟。

验证

1. 创建 pod，并使用 VRF 实例将其分配给额外网络：

   a. 创建定义 Pod 资源的 YAML 文件：

   pod-additional-net.yaml 文件示例

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-additional-net
 annotations:
   k8s.v1.cni.cncf.io/networks: '[
     {
           "name": "test-network-1" 1
     }
 ]'
spec:
 containers:
 - name: example-pod-1
   command: ["/bin/bash", "-c", "sleep 9000000"]
   image: centos:8
```

   1 使用 VRF 实例指定额外网络的名称。

   b. 运行以下命令来创建 Pod 资源：

```
$ oc create -f pod-additional-net.yaml
```

   输出示例

```
pod/test-pod created
```

2. 验证 Pod 网络附加是否已连接到 VRF 额外网络。使用 pod 启动远程会话并运行以下命令：

```
$ ip vrf show
```

   输出示例

```
Name              Table
-----------------------
vrf-1             1001
```

3. **确认 VRF 接口是额外接口的控制器：**

```
$ ip link
```

输出示例

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
red state UP mode
```

第 25 章 硬件网络

25.1. 关于单根 I/O 虚拟化（SR-IOV）硬件网络

Single Root I/O 虚拟化 (SR-IOV) 规范是针对一类 PCI 设备分配的标准，可与多个 pod 共享单个设备。

通过 SR-IOV，您可以将主机节点上识别为物理功能 (PF) 的兼容网络设备分段为多个虚拟功能 (VF)。VF 和其它网络设备一样使用。该设备的 SR-IOV 网络设备驱动程序决定了如何公开容器中的 VF：

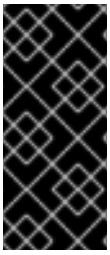
- netdevice 驱动程序： 容器 netns 中的常规内核网络设备
- vfio-pci 驱动程序： 挂载到容器中的字符设备

对于需要高带宽或低延迟的应用程序，您可以在裸机或 Red Hat OpenStack Platform(RHOSP)基础架构上安装 OpenShift Container Platform 集群上的额外网络使用 SR-IOV 网络设备。

您可以为 SR-IOV 网络配置多网络策略。对这个功能的支持是技术预览，SR-IOV 额外网络只支持内核 NIC。它们不支持 Data Plane Development Kit (DPDK) 应用程序。

注意

与 SR-IOV 网络相比，在 SR-IOV 网络中创建多网络策略可能无法为应用程序提供相同的性能。



重要

SR-IOV 网络的多网络策略只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以使用以下命令在节点上启用 SR-IOV：

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

25.1.1. 负责管理 SR-IOV 网络设备的组件

SR-IOV Network Operator 会创建和管理 SR-IOV 堆栈的组件。它执行以下功能：

- 编配 SR-IOV 网络设备的发现和管理
- 为 SR-IOV Container Network Interface（CNI）生成 NetworkAttachmentDefinition 自定义资源
- 创建和更新 SR-IOV 网络设备插件的配置
- 创建节点特定的 SriovNetworkNodeState 自定义资源
- 更新每个 SriovNetworkNodeState 自定义资源中的 spec.interfaces 字段

Operator 置备以下组件：

SR-IOV 网络配置守护进程

SR-IOV Network Operator 启动时部署在 worker 节点上的守护进程集。守护进程负责在集群中发现和初始化 SR-IOV 网络设备。

SR-IOV Network Operator Webhook

这是动态准入控制器 Webhook，用于验证 Operator 自定义资源，并为未设置的字段设置适当的默认值。

SR-IOV Network Resources Injector（网络资源注入器）

这是一个动态准入控制器 Webhook，它提供通过请求和限制为自定义网络资源（如 SR-IOV VF）应用 Kubernetes pod 规格的功能。SR-IOV 网络资源注入器只会将 **resource** 字段添加到 pod 中的第一个容器。

网络SR-IOV 网络设备插件

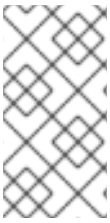
这个设备插件用于发现、公告并分配 SR-IOV 网络虚拟功能 (VF) 资源。在 Kubernetes 中使用设备插件能够利用有限的资源，这些资源通常为于物理设备中。设备插件可以使 Kubernetes 调度程序了解资源可用性，因此调度程序可以在具有足够资源的节点上调度 pod。

SR-IOV CNI 插件

SR-IOV CNI 插件会附加从 SR-IOV 网络设备插件中直接分配给 pod 的 VF 接口。

SR-IOV InfiniBand CNI 插件

附加从 SR-IOV 网络设备插件中直接分配给 pod 的 InfiniBand（IB）VF 接口的 CNI 插件。



注意

SR-IOV Network resources injector 和 SR-IOV Operator Webhook 会被默认启用，可通过编辑 **default SriovOperatorConfig** CR 来禁用。禁用 SR-IOV Network Operator Admission Controller Webhook 时要小心。您可以在特定情况下禁用 webhook，如故障排除，或者想要使用不支持的设备。

25.1.1.1. 支持的平台

在以下平台上支持 SR-IOV Network Operator：

- 裸机
- Red Hat OpenStack Platform(RHOSP)

25.1.1.2. 支持的设备

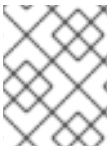
OpenShift Container Platform 支持以下网络接口控制器：

表 25.1. 支持的网络接口控制器

制造商	model	供应商 ID	设备 ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572

制造商	model	供应商 ID	设备 ID
Intel	X710 Backplane	8086	1581
Intel	X710 基本 T	8086	15ff
Intel	XL710	8086	1583
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Intel	E810-XXVDA4T	8086	1593
Mellanox	MT27700 系列 [ConnectX-4]	15b3	1013
Mellanox	MT27710 系列 [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 系列 [ConnectX-5]	15b3	1017
Mellanox	MT28880 系列 [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 系列 [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	Mellanox MT2910 系列 [ConnectX-7]	15b3	1021
Mellanox	ConnectX-6 NIC 模式中的 MT42822 BlueField-2	15b3	a2d6
Pensando [1]	用于 ionic 驱动程序的 DSC-25 双端口 25G 分布式服务卡	0x1dd8	0x1002
Pensando [1]	用于 ionic 驱动程序的 DSC-100 双端口 100G 分布式服务卡	0x1dd8	0x1003
Silicom	STS 系列	8086	1591

1. 支持 OpenShift SR-IOV，但在使用 SR-IOV 时，您必须使用 SR-IOV CNI 配置文件设置静态的虚拟功能(VF)介质访问控制(MAC)地址。



注意

有关支持的卡和兼容的 OpenShift Container Platform 版本的最新列表，请参阅 Openshift Single Root I/O Virtualization(SR-IOV)和 PTP 硬件网络支持列表。

25.1.1.3. 自动发现 SR-IOV 网络设备

SR-IOV Network Operator 将搜索集群以获取 worker 节点上的 SR-IOV 功能网络设备。Operator 会为每个提供兼容 SR-IOV 网络设备的 worker 节点创建并更新一个 SriovNetworkNodeState 自定义资源(CR)。

为 CR 分配了与 worker 节点相同的名称。status.interfaces 列表提供有关节点上网络设备的信息。

重要

不要修改 SriovNetworkNodeState 对象。Operator 会自动创建和管理这些资源。

25.1.1.3.1. SriovNetworkNodeState 对象示例

以下 YAML 是由 SR-IOV Network Operator 创建的 SriovNetworkNodeState 对象的示例：

一个 SriovNetworkNodeState 对象

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 (1)
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: (2)
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
```

```
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded
```

1 name 字段的值与 worker 节点的名称相同。

2 interfaces 小节包括 Operator 在 worker 节点上发现的所有 SR-IOV 设备列表。

25.1.1.4. 在 pod 中使用虚拟功能的示例

您可以在附加了 SR-IOV VF 的 pod 中运行远程直接内存访问 (RDMA) 或 Data Plane Development Kit (DPDK) 应用程序。

本示例演示了在 RDMA 模式中使用虚拟功能 (VF) 的 pod：

使用 RDMA 模式的 Pod 规格

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
    command: ["sleep", "infinity"]
```

以下示例演示了在 DPDK 模式中使用 VF 的 pod:

使用 DPDK 模式的 Pod 规格

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
      requests:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

25.1.1.5. 用于容器应用程序的 DPDK 库

一个可选的库 app-netutil 提供了几个 API 方法，用于从该 pod 中运行的容器内收集 pod 的网络信息。

此库可以将 SR-IOV 虚拟功能（VF）集成到 Data Plane Development Kit（DPDK）模式中。该程序库提供 Golang API 和 C API。

当前，采用三个 API 方法：

GetCPUInfo()

此函数决定了哪些 CPU 可供容器使用并返回相关列表。

GetHugepages()

此函数决定了每个容器在 Pod spec 中请求的巨页内存量并返回相应的值。

GetInterfaces()

此函数决定了容器中的一组接口，并返回相关的列表。返回值包括每个接口的接口类型和特定于类型的数据。

库的存储库包括用于构建容器镜像 dpdk-app-centos 的示例 Dockerfile。根据容器集规格中的环境变

量，容器镜像可以运行以下 DPDK 示例应用之一： l2fwd、l3wd 或 testpmd。容器镜像提供了一个将 app-netutil 库集成到容器镜像本身的示例。库也可以集成到 init 容器中。init 容器可以收集所需的数据，并将数据传递给现有的 DPDK 工作负载。

25.1.1.6. Downward API 的巨页资源注入

当 pod 规格包含巨页的资源请求或限制时，Network Resources Injector 会自动在 pod 规格中添加 Downward API 字段，以便为容器提供巨页信息。

Network Resources Injector 添加一个名为 podnetinfo 的卷，并挂载到 pod 中的每个容器的 /etc/podnetinfo。卷使用 Downward API，并包含一个用于大页面请求和限制的文件。文件命名规则如下：

- /etc/podnetinfo/hugepages_1G_request_<container-name>
- /etc/podnetinfo/hugepages_1G_limit_<container-name>
- /etc/podnetinfo/hugepages_2M_request_<container-name>
- /etc/podnetinfo/hugepages_2M_limit_<container-name>

上一个列表中指定的路径与 app-netutil 库兼容。默认情况下，该库配置为搜索 /etc/podnetinfo 目录中的资源信息。如果您选择自己手动指定 Downward API 路径项目，app-netutil 库除上一个列表中的路径外还会搜索以下路径。

- /etc/podnetinfo/hugepages_request
- /etc/podnetinfo/hugepages_limit
- /etc/podnetinfo/hugepages_1G_request
- /etc/podnetinfo/hugepages_1G_limit
- /etc/podnetinfo/hugepages_2M_request
- /etc/podnetinfo/hugepages_2M_limit

与 Network Resources Injector 可以创建的路径一样，以上列表中的路径可以选择以一个 _<container-name> 后缀结尾。

25.1.2. 其他资源

- 配置多网络策略

25.1.3. 后续步骤

- 安装 SR-IOV Network Operator
- 可选：配置 SR-IOV Network Operator
- 配置 SR-IOV 网络设备
- 如果使用 OpenShift Virtualization： 将虚拟机连接到 SR-IOV 网络
- 配置 SR-IOV 网络附加

- 将 pod 添加到额外网络

25.2. 安装 SR-IOV NETWORK OPERATOR

您可以在集群上安装单根 I/O 虚拟化（SR-IOV）网络 Operator，以管理 SR-IOV 网络设备和网络附加。

25.2.1. 安装 SR-IOV Network Operator

作为集群管理员，您可以使用 OpenShift Container Platform CLI 或 Web 控制台安装单根 I/O 虚拟化 (SR-IOV) Network Operator。

25.2.1.1. CLI：安装 SR-IOV Network Operator

作为集群管理员，您可以使用 CLI 安装 Operator。

先决条件

- 在裸机环境中安装的集群，其中的节点带有支持 SR-IOV 的硬件。
- 安装 OpenShift CLI（oc）。
- 具有 cluster-admin 特权的帐户。

流程

1. 要创建 openshift-sriov-network-operator 命名空间，输入以下命令：

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2. 运行以下命令来创建 OperatorGroup CR：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
EOF
```

3. 要为 SR-IOV Network Operator 创建 Subscription CR，输入以下命令：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
  metadata:
    name: sriov-network-operator-subscription
    namespace: openshift-sriov-network-operator
  spec:
    channel: stable
    name: sriov-network-operator
    source: redhat-operators
    sourceNamespace: openshift-marketplace
  EOF
```

4. 要验证是否已安装 Operator，请输入以下命令：

```
$ oc get csv -n openshift-sriov-network-operator \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

输出示例

```
Name                                         Phase
sriov-network-operator.4.16.0-202310121402   Succeeded
```

25.2.1.2. web 控制台：安装 SR-IOV Network Operator

作为集群管理员，您可以使用 Web 控制台安装 Operator。

先决条件

- 在裸机环境中安装的集群，其中的节点带有支持 SR-IOV 的硬件。
- 安装 OpenShift CLI（oc）。
- 具有 cluster-admin 特权的帐户。

流程

1. 安装 SR-IOV Network Operator：
   a. 在 OpenShift Container Platform Web 控制台中，点击 Operators → OperatorHub。
   b. 从可用的 Operators 列表中选择 SR-IOV Network Operator，然后点击 Install。
   c. 在 Install Operator 页面中，在 Installed Namespace 下选择 Operator recommended Namespace。
   d. 点 Install。
2. 验证 SR-IOV Network Operator 是否已成功安装：
   a. 导航到 Operators → Installed Operators 页面。
   b. 确保 SR-IOV Network Operator 在 openshift-sriov-network-operator 项目中列出，状态为 InstallSucceeded。

注意

在安装过程中，Operator 可能会显示 Failed 状态。如果安装过程结束后有 InstallSucceeded 信息，您可以忽略这个 Failed 信息。

如果 Operator 没有被成功安装，请按照以下步骤进行故障排除：

- 检查 Operator Subscriptions 和 Install Plans 选项卡中的 Status 项中是否有任何错误。
- 进入 Workloads → Pods 页面，在 openshift-sriov-network-operator 项目中检查 pod 的日志。
- 检查 YAML 文件的命名空间。如果缺少注解，您可以使用以下命令将注解 workload.openshift.io/allowed=management 添加到 Operator 命名空间中：

```
$ oc annotate ns/openshift-sriov-network-operator
workload.openshift.io/allowed=management
```

注意

对于单节点 OpenShift 集群，命名空间需要注解 workload.openshift.io/allowed=management。

25.2.2. 后续步骤

- 配置 SR-IOV Network Operator

25.3. 配置 SR-IOV NETWORK OPERATOR

Single Root I/O Virtualization（SR-IOV）Network Operator 管理集群中的 SR-IOV 网络设备和网络附加。

25.3.1. 配置 SR-IOV Network Operator

- 创建一个 SriovOperatorConfig 自定义资源 (CR) 以部署所有 SR-IOV Operator 组件：
  a. 使用以下 YAML 创建名为 sriovOperatorConfig.yaml 的文件：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: false
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
```

注意

SriovOperatorConfig 资源的唯一有效名称是 default，它必须位于部署 Operator 的命名空间中。

b. 运行以下命令来创建资源：

```
$ oc apply -f sriovOperatorConfig.yaml
```

25.3.1.1. SR-IOV Network Operator 配置自定义资源

sriovoperatorconfig 自定义资源的字段在下表中描述：

表 25.2. SR-IOV Network Operator 配置自定义资源

字段	类型	描述
metadata.name	string	指定 SR-IOV Network Operator 实例的名称。默认值为 default。不要设置不同的值。
metadata.namespace	string	指定 SR-IOV Network Operator 实例的命名空间。默认值为 openshift-sriov-network-operator。不要设置不同的值。
spec.configDaemonNodeSelector	string	指定在所选节点上调度 SR-IOV 网络配置守护进程的节点选择。默认情况下，此字段没有设置，Operator 会在 worker 节点上部署 SR-IOV 网络配置守护进程集。
spec.disableDrain	布尔值	指定是否禁用节点排空过程，或者在应用新策略在节点上配置 NIC 时启用节点排空过程。将此字段设置为 true 可促进软件开发，并在单一节点上安装 OpenShift Container Platform。默认情况下不设置此字段。 对于单节点集群，在安装 Operator 后将此字段设置为 true。此字段必须保持设为 true。
spec.enableInjector	布尔值	指定是否启用或禁用 Network Resources Injector 守护进程集。默认情况下，此字段设置为 true。
spec.enableOperatorWebhook	布尔值	指定是否启用或禁用 Operator Admission Controller webhook 守护进程集。默认情况下，此字段设置为 true。
spec.logLevel	整数	指定 Operator 的日志详细程度。设置为 0 以仅显示基本日志。设置为 2，以显示所有可用的日志。默认情况下，此字段设置为 2。

25.3.1.2. 关于 Network Resources Injector（网络资源注入器）

Network Resources Injector 是一个 Kubernetes Dynamic Admission Controller 应用。它提供以下功能：

- 根据 SR-IOV 网络附加定义注解，对 Pod 规格中的资源请求和限值进行修改，以添加 SR-IOV 资源名称。
- 使用 Downward API 卷修改 pod 规格，以公开 pod 注解、标签和巨页请求和限制。在 pod 中运行的容器可以作为 /etc/podnetinfo 路径下的文件来访问公开的信息。

默认情况下 Network Resources Injector 由 SR-IOV Network Operator 应用，并作为守护进程集在所

默认情况下，Network Resources Injector 由 SR-IOV Network Operator 启用，并作为守护进程集在所有 control plane 节点上运行。以下是在具有三个 control plane 节点的集群中运行的 Network Resources Injector pod 示例：

```
$ oc get pods -n openshift-sriov-network-operator
```

输出示例

```
NAME                                     READY   STATUS    RESTARTS   AGE
network-resources-injector-5cz5p         1/1     Running   0          10m
network-resources-injector-dwqpx         1/1     Running   0          10m
network-resources-injector-lktz5         1/1     Running   0          10m
```

25.3.1.3. 关于 SR-IOV Network Operator 准入控制器 Webhook

SR-IOV Network Operator Admission Controller Webhook 是一个 Kubernetes Dynamic Admission Controller 应用程序。它提供以下功能：

- 在创建或更新时，验证 **SriovNetworkNodePolicy** CR。
- 修改 **SriovNetworkNodePolicy** CR，在创建或更新 CR 时为**priority** 和 **deviceType** 项设置默认值。

默认情况下，SR-IOV Network Operator Admission Controller Webhook 由 Operator 启用，并作为守护进程集在所有 control plane 节点上运行。

注意

禁用 SR-IOV Network Operator Admission Controller Webhook 时要小心。您可以在特定情况下禁用 webhook，如故障排除，或者想要使用不支持的设备。有关配置不支持的设备的详情，请参考将 SR-IOV Network Operator 配置为使用不支持的 NIC。

以下是在具有三个 control plane 节点的集群中运行的 Operator Admission Controller webhook pod 的示例：

```
$ oc get pods -n openshift-sriov-network-operator
```

输出示例

```
NAME                                     READY   STATUS    RESTARTS   AGE
operator-webhook-9jkw6                   1/1     Running   0          16m
operator-webhook-kbr5p                   1/1     Running   0          16m
operator-webhook-rpfrl                   1/1     Running   0          16m
```

25.3.1.4. 关于自定义节点选择器

SR-IOV 网络配置守护进程在集群节点上发现并配置 SR-IOV 网络设备。默认情况下，它将部署到集群中的所有 worker 节点。您可以使用节点标签指定 SR-IOV 网络配置守护进程在哪些节点上运行。

25.3.1.5. 禁用或启用网络资源注入器

要禁用或启用默认启用的网络资源注入器，请完成以下步骤。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 您必须已安装了 SR-IOV Network Operator。

流程

- 设置 enableInjector 字段。将 <value> 替换为 false 来禁用这个功能；或替换为 true 来启用这个功能。

```
$ oc patch sriovoperatorconfig default \
  --type=merge -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableInjector": <value> } }'
```

提示

您还可以应用以下 YAML 来更新 Operator：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
```

25.3.1.6. 禁用或启用 SR-IOV Network Operator 准入控制器 Webhook

要禁用或启用默认启用的准入控制器 Webhook，请完成以下步骤。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 您必须已安装了 SR-IOV Network Operator。

流程

- 设置 enableOperatorWebhook 字段。将 <value> 替换为 false 来禁用这个功能；或替换为 true 来启用这个功能：

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

提示

您还可以应用以下 YAML 来更新 Operator：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

25.3.1.7. 为 SR-IOV 网络配置守护进程配置自定义 NodeSelector

SR-IOV 网络配置守护进程在集群节点上发现并配置 SR-IOV 网络设备。默认情况下，它将部署到集群中的所有 worker 节点。您可以使用节点标签指定 SR-IOV 网络配置守护进程在哪些节点上运行。

要指定部署了 SR-IOV 网络配置守护进程的节点，请完成以下步骤。

重要

当您更新 configDaemonNodeSelector 字段时，SR-IOV 网络配置守护进程会在所选节点中重新创建。在重新创建守护进程时，集群用户无法应用任何新的 SR-IOV 网络节点策略或创建新的 SR-IOV Pod。

流程

- 要为 Operator 更新节点选择器，请输入以下命令：

```
$ oc patch sriovoperatorconfig default --type=json \
  -n openshift-sriov-network-operator \
  --patch '[{
      "op": "replace",
      "path": "/spec/configDaemonNodeSelector",
      "value": {<node_label>}
    }]'
```

将 <node_label> 替换为要应用的标签，如下例中："node-role.kubernetes.io/worker": ""。

提示

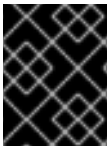
您还可以应用以下 YAML 来更新 Operator：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
```

25.3.1.8. 为单一节点安装配置 SR-IOV Network Operator

默认情况下，SR-IOV Network Operator 会在每次策略更改前从节点排空工作负载。Operator 会执行这个操作，以确保在重新配置前没有使用虚拟功能的工作负载。

对于在单一节点上安装，没有其他节点来接收工作负载。因此，Operator 不得配置为从单一节点排空工作负载。



重要

执行以下步骤禁用排空工作负载后，您必须删除所有使用 SR-IOV 网络接口的工作负载，然后才能更改任何 SR-IOV 网络节点策略。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 您必须已安装了 SR-IOV Network Operator。

流程

- 要将 disableDrain 字段设置为 true，并将 configDaemonNodeSelector 字段设置为 node-role.kubernetes.io/master: ""，请输入以下命令：

```
$ oc patch sriovoperatorconfig default --type=merge -n openshift-sriov-network-operator --patch '{ "spec": { "disableDrain": true, "configDaemonNodeSelector": { "node-role.kubernetes.io/master": "" } } }'
```

提示

您还可以应用以下 YAML 来更新 Operator：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: true
  configDaemonNodeSelector:
   node-role.kubernetes.io/master: ""
```

25.3.1.9. 为托管 control plane 部署 SR-IOV Operator

重要

在 AWS 平台上托管的 control plane 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

配置和部署托管服务集群后，您可以在托管集群中创建 SR-IOV Operator 订阅。SR-IOV pod 在 worker 机器上运行而不是在 control plane 上运行。

先决条件

您必须在 AWS 上配置和部署托管集群。如需更多信息，请参阅在 AWS 上配置托管集群（技术预览）。

流程

1. 创建命名空间和 Operator 组：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
```

2. 创建 SR-IOV Operator 的订阅：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

验证

1. 要验证 SR-IOV Operator 是否已就绪，请运行以下命令并查看生成的输出：

```
$ oc get csv -n openshift-sriov-network-operator
```

输出示例

```
NAME                                         DISPLAY                   VERSION            REPLACES
PHASE
sriov-network-operator.4.16.0-202211021237   SR-IOV Network Operator   4.16.0-
202211021237   sriov-network-operator.4.16.0-202210290517   Succeeded
```

2. 要验证 SR-IOV pod 是否已部署，请运行以下命令：

```
$ oc get pods -n openshift-sriov-network-operator
```

25.3.2. 后续步骤

- 配置 SR-IOV 网络设备

25.4. 配置 SR-IOV 网络设备

您可以在集群中配置单一根 I/O 虚拟化（SR-IOV）设备。

25.4.1. SR-IOV 网络节点配置对象

您可以通过创建 SR-IOV 网络节点策略来为节点指定 SR-IOV 网络设备配置。策略的 API 对象是 sriovnetwork.openshift.io API 组的一部分。

以下 YAML 描述了 SR-IOV 网络节点策略：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  needVhostNet: false 7
  numVfs: <num> 8
  externallyManaged: false 9
  nicSelector: 10
    vendor: "<vendor_code>" 11
    deviceID: "<device_id>" 12
    pfNames: ["<pf_name>", ...] 13
    rootDevices: ["<pci_bus_id>", ...] 14
    netFilter: "<filter_string>" 15
  deviceType: <device_type> 16
  isRdma: false 17
  linkType: <link_type> 18
  eSwitchMode: "switchdev" 19
  excludeTopology: false 20
```

1. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。

在指定名称时，请确保在 resourceName 中使用接受的语法表达式 ^[a-zA-Z0-9_]+$。

4 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。

重要

SR-IOV Network Operator 按顺序将节点网络配置策略应用到节点。在应用节点网络配置策略前，SR-IOV Network Operator 会检查节点的机器配置池(MCP)是否处于不健康状态，如 Degraded 或 Updating。如果节点处于不健康的 MCP，将节点网络配置策略应用到集群中的所有目标节点的过程会被暂停，直到 MCP 返回健康状态。

为了避免处于不健康的 MCP 的节点阻止将节点网络配置策略应用到其他节点，包括处于其他 MCP 的节点，您必须为每个 MCP 创建单独的节点网络配置策略。

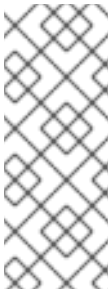
5 可选： priority 是一个 0 到 99 之间的整数。较小的值具有更高的优先级。例如，优先级 10 是高于优先级 99。默认值为 99。

6 可选：虚拟功能的最大传输单元（MTU）。最大 MTU 值可能因不同的网络接口控制器（NIC）型号而有所不同。

7 可选：将 needVhostNet 设置为 true，以在 pod 中挂载/dev/vhost-net 设备。使用挂载的 /dev/vhost-net 设备及 Data Plane Development Kit (DPDK) 将流量转发到内核网络堆栈。

8 为 SR-IOV 物理网络设备创建的虚拟功能(（VF）的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 128。

9 将 externallyManaged 设置为 true，以允许 SR-IOV Network Operator 使用所有或外部管理虚拟功能 (VF) 的子集并将它们附加到 pod。将值设为 false 时，SR-IOV Network Operator 管理并配置所有分配的 VF。



注意

当 externallyManaged 设置为 true 时，您必须在应用策略前创建虚拟功能 (VF)。如果没有设置，webhook 将阻断请求。如果 externallyManaged 被设置为 false，SR-IOV Network Operator 会处理 VF 的创建和管理，包括根据需要重置 VF。因此，要在主机系统上使用 VF，必须手动创建，externallyManaged 必须设置为 true，因此 SR-IOV Network Operator 不会对 PF 以及在策略 nicSelector 中定义的 VF 执行任何操作。

10 NIC 选择器标识要配置的 Operator 的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。

如果指定了 rootDevices，则必须同时为 vendor、 deviceID 或 pfNames 指定一个值。如果同时指定了 pfNames 和 rootDevices，请确保它们引用同一设备。如果您为 netFilter 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。

11 可选： SR-IOV 网络设备的厂商十六进制代码。允许的值只能是 8086 和 15b3。

12 可选： SR-IOV 网络设备的设备十六进制代码。例如，101b 是 Mellanox ConnectX-6 设备的设备 ID。

13 可选：该设备的一个或多个物理功能（PF）名称的数组。

14 可选：用于该设备的 PF 的一个或多个 PCI 总线地址的数组。使用以下格式提供地址：0000:02:00.1。

15 可选：特定平台的网络过滤器。唯一支持的平台是 Red Hat OpenStack Platform（RHOSP）。可接受的值具有以下格式： openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx。将 xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx 替换为来自 /var/config/openstack/latest/network_data.json 元数据文件的值。

16 可选：虚拟功能的驱动程序类型。允许的值只能是 netdevice 和 vfio-pci。默认值为 netdevice。

对于裸机节点上的 DPDK 模式的 Mellanox NIC，请使用 netdevice 驱动程序类型，并将 isRdma 设置为 true。

17 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 false。

如果 isRdma 参数设为 true，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。

将 isRdma 设置为 true，并将 needVhostNet 设置为 true 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。

18 可选：VF 的链接类型。默认值为 eth （以太网）。在 InfiniBand 中将这个值改为 'ib'。

当将 linkType 设置为 ib 时，SR-IOV Network Operator Webhook 会自动将isRdma 设置为 true。当将 linkType 设定为 ib 时，deviceType 不应该被设置为 vfio-pci。

不要为 SriovNetworkNodePolicy 将 linkType 设置为 eth，因为这可能会导致设备插件报告的可用设备数量不正确。

19 可选： 要启用硬件卸载，必须将 eSwitchMode 字段设置为 "switchdev"。

20 可选：要排除将一个 SR-IOV 网络资源的 NUMA 节点广告到拓扑管理器，将值设为 true。默认值为 false。

25.4.1.1. SR-IOV 网络节点配置示例

以下示例描述了 InfiniBand 设备的配置：

InfiniBand 设备的配置示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-ib-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: ibnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
    rootDevices:
```

```
    - "0000:19:00.0"
  linkType: ib
  isRdma: true
```

以下示例描述了 RHOSP 虚拟机中的 SR-IOV 网络设备配置：

虚拟机中的 SR-IOV 设备配置示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-sriov-net-openstack-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 (1)
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
    netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" (2)
```

(1) 在为虚拟机配置节点网络策略时，numVfs 字段始终设置为 1。

(2) 当虚拟机在 RHOSP 上部署时，netFilter 字段必须引用网络 ID。netFilter 的有效值来自 SriovNetworkNodeState 对象。

25.4.1.2. SR-IOV 设备的虚拟功能 (VF) 分区

在某些情况下，您可能想要将同一个物理功能 (PF) 的虚拟功能 (VF) 分成多个资源池。例如：您可能想要某些 VF 使用默认驱动程序载入，而其他的 VF 负载使用 vfio-pci 驱动程序。在这样的部署中，您可以使用SriovNetworkNodePolicy 自定义资源 (CR) 中的 pfNames 选项器（selector）来为池指定 VF 的范围，其格式为：<pfname>#<first_vf>-<last_vf>。

例如，以下 YAML 显示名为 netpf0 的、带有 VF 2 到 7 的接口的选择器：

```
pfNames: ["netpf0#2-7"]
```

- netpf0 是 PF 接口名称。
- 2 是包含在范围内的第一个 VF 索引（基于 0）。
- 7 是包含在范围内的最后一个 VF 索引（基于 0）。

如果满足以下要求，您可以使用不同的策略 CR 从同一 PF 中选择 VF:

- 选择相同 PF 的不同策略的 numVfs 值必须相同。
- VF 索引范围是从 0 到 <numVfs>-1 之间。例如，如果您有一个策略，它的 numVfs 被设置为 8，则 <first_vf> 的值不能小于 0，<last_vf> 的值不能大于 7。
- 不同策略中的 VF 范围不得互相重叠。

- <first_vf> 不能大于 <last_vf>。

以下示例演示了 SR-IOV 设备的 NIC 分区。

策略 policy-net-1 定义了一个资源池 net-1，其中包含带有默认 VF 驱动的 PF netpf0 的 VF 0 。策略 policy-net-1-dpdk 定义了一个资源池 net-1-dpdk，其中包含带有 vfio VF 驱动程序的 PF netpf0 的 VF 8 到 15。

策略 policy-net-1:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

策略 policy-net-1-dpdk:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci
```

验证接口是否已成功分区

运行以下命令，确认 SR-IOV 设备的接口分区到虚拟功能(VF)。

```
$ ip link show <interface> 1
```

1 将 <interface> 替换为您在分区为 SR-IOV 设备的 VF 时指定的接口，如ens3f1。

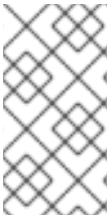
输出示例

```
5: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 3c:fd:fe:d1:bc:01 brd ff:ff:ff:ff:ff:ff
```

```
vf 0    link/ether 5a:e7:88:25:ea:a0 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 1    link/ether 3e:1d:36:d7:3d:49 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 2    link/ether ce:09:56:97:df:f9 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 3    link/ether 5e:91:cf:88:d1:38 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
vf 4    link/ether e6:06:a1:96:2f:de brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust
off
```

25.4.2. 配置 SR-IOV 网络设备

SR-IOV Network Operator 把 SriovNetworkNodePolicy.sriovnetwork.openshift.io CRD 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。



注意

在应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。

它可能需要几分钟时间来应用配置更改。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。
- 集群中有足够的可用节点，用于处理从排空节点中驱除的工作负载。
- 您还没有为 SR-IOV 网络设备配置选择任何 control plane 节点。

流程

1. 创建一个 SriovNetworkNodePolicy 对象，然后在 <name>-sriov-node-network.yaml 文件中保存 YAML。使用配置的实际名称替换 <name>。
2. 可选：将 SR-IOV 功能的集群节点标记为 SriovNetworkNodePolicy.Spec.NodeSelector （如果它们还没有标记）。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。
3. 创建 SriovNetworkNodePolicy 对象：

   ```
   $ oc create -f <name>-sriov-node-network.yaml
   ```

   其中 <name> 指定这个配置的名称。

   在应用配置更新后，sriov-network-operator 命名空间中的所有 Pod 都会变为 Running 状态。
4. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 <node_name> 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

其他资源

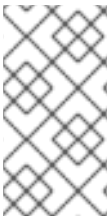
- 了解如何更新节点 上的标签。

25.4.2.1. 在 SR-IOV 网络策略更新过程中配置并行节点排空

默认情况下，SR-IOV Network Operator 会在每次策略更改前从节点排空工作负载。Operator 一次执行这个操作，一个节点以确保没有工作负载受到重新配置的影响。

在大型集群中，按顺序排空节点可能非常耗时，需要几小时甚至几天。在时间敏感的环境中，您可以在 SriovNetworkPoolConfig 自定义资源 (CR) 中启用并行节点排空，以更快地推出 SR-IOV 网络配置。

要配置并行排空，请使用 SriovNetworkPoolConfig CR 创建节点池。然后，您可以在池中添加节点，并在 Operator 可以并行排空的池中定义最大节点数。使用这个方法，您可以启用并行排空来更快地重新配置，同时确保池中仍有足够的节点来处理任何正在运行的工作负载。



注意

节点只能属于一个 SR-IOV 网络池配置。如果节点不是池的一部分，则会将其添加到虚拟（默认）中，该池配置为仅排空一个节点。

节点可能会在排空过程中重启。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 SR-IOV Network Operator。
- 节点具有支持 SR-IOV 的硬件。

流程

1. 创建一个 SriovNetworkPoolConfig 资源：
   a. 创建一个定义 SriovNetworkPoolConfig 资源的 YAML 文件：

      sriov-nw-pool.yaml 文件示例

```
apiVersion: v1
kind: SriovNetworkPoolConfig
metadata:
  name: pool-1 1
  namespace: openshift-sriov-network-operator 2
spec:
  maxUnavailable: 2 3
  nodeSelector: 4
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

1. 指定 SriovNetworkPoolConfig 对象的名称。
2. 指定安装 SR-IOV Network Operator 的命名空间。
3. 为在更新过程中可用的节点指定一个整数或百分比值。例如，如果您有 10 个节点，并且将最大不可用设置为 2，那么可以随时并行排空 2 个节点，保留 8 个节点来处理工作负载。
4. 使用节点选择器指定要添加池的节点。本例将具有 worker 角色的所有节点添加到池中。

b. 运行以下命令来创建 SriovNetworkPoolConfig 资源：

```
$ oc create -f sriov-nw-pool.yaml
```

2. 运行以下命令来创建 sriov-test 命名空间：

```
$ oc create namespace sriov-test
```

3. 创建一个 SriovNetworkNodePolicy 资源：

a. 创建一个定义 SriovNetworkNodePolicy 资源的 YAML 文件：

sriov-node-policy.yaml 文件示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames: ["ens1"]
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  numVfs: 5
  priority: 99
  resourceName: sriov_nic_1
```

b. 运行以下命令来创建 SriovNetworkNodePolicy 资源：

```
$ oc create -f sriov-node-policy.yaml
```

4. 创建一个 SriovNetwork 资源：

a. 创建一个定义 SriovNetwork 资源的 YAML 文件：

sriov-network.yaml 文件示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nic-1
```

```
  namespace: openshift-sriov-network-operator
spec:
  linkState: auto
  networkNamespace: sriov-test
  resourceName: sriov_nic_1
  capabilities: '{ "mac": true, "ips": true }'
  ipam: '{ "type": "static" }'
```

b. 运行以下命令来创建 SriovNetwork 资源：

```
$ oc create -f sriov-network.yaml
```

验证

- 运行以下命令，查看您创建的节点池：

```
$ oc get sriovNetworkpoolConfig -n openshift-sriov-network-operator
```

输出示例

```
NAME     AGE
pool-1   67s 1
```

1 在本例中，pool-1 包含具有 worker 角色的所有节点。

要使用上述流程中的示例场景演示节点排空过程，请完成以下步骤：

1. 更新 SriovNetworkNodePolicy 资源中的虚拟功能数量，以触发集群中的工作负载排空：

```
$ oc patch SriovNetworkNodePolicy sriov-nic-1 -n openshift-sriov-network-operator --
type merge -p '{"spec": {"numVfs": 4}}'
```

2. 运行以下命令监控目标集群上的排空状态：

```
$ oc get sriovNetworkNodeState -n openshift-sriov-network-operator
```

输出示例

```
NAMESPACE                   NAME      SYNC STATUS   DESIRED SYNC STATE
CURRENT SYNC STATE   AGE
openshift-sriov-network-operator   worker-0   InProgress    Drain_Required
DrainComplete        3d10h
openshift-sriov-network-operator   worker-1   InProgress    Drain_Required
DrainComplete        3d10h
```

当排空过程完成后，SYNC STATUS 变为 Succeeded，DESIRED SYNC STATE 和 CURRENT SYNC STATE 值返回到 IDLE。

输出示例

```
NAMESPACE                   NAME      SYNC STATUS   DESIRED SYNC STATE
```

```
CURRENT SYNC STATE   AGE
openshift-sriov-network-operator   worker-0   Succeeded   Idle          Idle
3d10h
openshift-sriov-network-operator   worker-1   Succeeded   Idle          Idle
3d10h
```

25.4.3. SR-IOV 配置故障排除

在进行了配置 SR-IOV 网络设备的步骤后，以下部分会处理一些错误条件。

要显示节点状态，请运行以下命令：

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

其中： <node_name> 指定带有 SR-IOV 网络设备的节点名称。

错误输出： 无法分配内存

```
"lastSyncError": "write /sys/bus/pci/devices/0000:3b:00.1/sriov_numvfs: cannot allocate
memory"
```

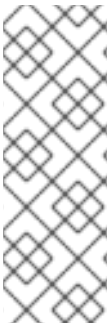
当节点表示无法分配内存时，检查以下项目：

- 确认在 BIOS 中为节点启用了全局 SR-IOV 设置。
- 确认在 BIOS 中为该节点启用了 VT-d。

25.4.4. 将 SR-IOV 网络分配给 VRF

作为集群管理员，您可以使用 CNI VRF 插件为 VRF 域分配 SR-IOV 网络接口。

要做到这一点，将 VRF 配置添加到 SriovNetwork 资源的可选 metaPlugins 参数中。



注意

使用 VRF 的应用程序需要绑定到特定设备。通常的用法是在套接字中使用 SO_BINDTODEVICE 选项。SO_BINDTODEVICE 将套接字绑定到在传递接口名称中指定的设备，例如 eth1。要使用 SO_BINDTODEVICE，应用程序必须具有 CAP_NET_RAW 功能。

OpenShift Container Platform pod 不支持通过 ip vrf exec 命令使用 VRF。要使用 VRF，将应用程序直接绑定到 VRF 接口。

25.4.4.1. 使用 CNI VRF 插件创建额外的 SR-IOV 网络附加

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 NetworkAttachmentDefinition 自定义资源（CR）。

注意

不要编辑 SR-IOV Network Operator 所管理的 NetworkAttachmentDefinition 自定义资源。这样做可能会破坏额外网络上的网络流量。

要使用 CNI VRF 插件创建额外的 SR-IOV 网络附加，请执行以下步骤。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为额外 SR-IOV 网络附加创建 SriovNetwork 自定义资源 (CR) 并插入metaPlugins 配置，如下例所示。将 YAML 保存为文件 sriov-network-attachment.yaml。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: example-network
  namespace: additional-sriov-network-1
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  vlan: 0
  resourceName: intelnics
  metaPlugins : |
    {
      "type": "vrf", 1
      "vrfname": "example-vrf-name" 2
    }
```

1. type 必须设为 vrf。
2. vrfname 是接口分配的 VRF 的名称。如果 pod 中不存在，则创建它。

2. 创建 SriovNetwork 资源：

```
$ oc create -f sriov-network-attachment.yaml
```

验证 NetworkAttachmentDefinition CR 是否已成功创建

- 运行以下命令，确认 SR-IOV Network Operator 创建了 NetworkAttachmentDefinition CR。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 将 <namespace> 替换为您在配置网络附加时指定的命名空间，如 additional-sriov-network-1。

输出示例

```
NAME                         AGE
additional-sriov-network-1     14m
```

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

验证额外 SR-IOV 网络附加是否成功

要验证 VRF CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

1. 创建使用 VRF CNI 的 SR-IOV 网络。
2. 将网络分配给 pod。
3. 验证 pod 网络附加是否已连接到 SR-IOV 额外网络。远程 shell 到 pod 并运行以下命令：

```
$ ip vrf show
```

输出示例

```
Name              Table
-----------------------
red                 10
```

4. 确认 VRF 接口是从属接口的主接口：

```
$ ip link
```

输出示例

```
...
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
red state UP mode
...
```

25.4.5. 为 NUMA 感知调度排除 SR-IOV 网络拓扑

在 NUMA 感知 pod 调度过程中，可以排除将 SR-IOV 网络的 Non-Uniform Memory Access (NUMA) 节点广告到拓扑管理器，以便实现更灵活的 SR-IOV 网络部署。

在某些情况下，为在单个 NUMA 节点上的一个 pod 最大化 CPU 和内存资源是一个优先操作。如果没有为 Topology Manager 提供有关 pod 的 SR-IOV 网络资源的 NUMA 节点的提示，拓扑管理器可能会将 SR-IOV 网络资源和 pod CPU 和内存资源部署到不同的 NUMA 节点。这可能会添加到网络延迟，因为需要在不同 NUMA 节点之间进行数据传输。但是，当工作负载需要最佳 CPU 和内存性能时，这是可以接受的。

例如，有一个计算节点 compute-1，它有两个 NUMA 节点：numa0 和 numa1。启用了 SR-IOV 的 NIC

存在于 numa0 上。可用于 pod 调度的 CPU 仅存在于numa1 上。通过将 excludeTopology 规格设置为 true，拓扑管理器可将 pod 的 CPU 和内存资源分配给numa1，并可将同一 pod 的 SR-IOV 网络资源分配给 numa0。只有将 excludeTopology 规格设置为 true 时，才能实现。否则，拓扑管理器会尝试将所有资源放在同一 NUMA 节点上。

25.4.5.1. 排除 NUMA 感知调度的 SR-IOV 网络拓扑

要将 SR-IOV 网络资源的 Non-Uniform Memory Access (NUMA)节点排除到拓扑管理器，您可以在 SriovNetworkNodePolicy 自定义资源中配置 excludeTopology 规格。在 NUMA 感知 pod 调度过程中，使用此配置来实现更灵活的 SR-IOV 网络部署。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您已将 CPU Manager 策略配置为 static。有关 CPU Manager 的更多信息，请参阅*附加资源*部分。
- 您已将 Topology Manager 策略配置为 single-numa-node。
- 已安装 SR-IOV Network Operator。

流程

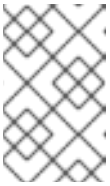
1. 创建 SriovNetworkNodePolicy CR：
   a. 将以下 YAML 保存到 sriov-network-node-policy.yaml 文件中，替换 YAML 中的值以匹配您的环境：

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <policy_name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 (1)
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  numVfs: <number_of_Vfs>
  nicSelector: (2)
    vendor: "<vendor_ID>"
    deviceID: "<device_ID>"
  deviceType: netdevice
  excludeTopology: true (3)
```

(1) SR-IOV 网络设备插件的资源名称。此 YAML 使用示例 resourceName 值。

(2) 使用 NIC 选择器识别要配置的 Operator 的设备。

(3) 要将 SR-IOV 网络资源的 NUMA 节点排除到拓扑管理器，请将值设为 true。默认值为 false。



注意

如果多个 SriovNetworkNodePolicy 资源都以同一 SR-IOV 网络资源为目标，则 SriovNetworkNodePolicy 资源必须具有与 excludeTopology 规格相同的值。否则，冲突策略将被拒绝。

b. **运行以下命令来创建 SriovNetworkNodePolicy 资源：**

```
$ oc create -f sriov-network-node-policy.yaml
```

输出示例

```
sriovnetworknodepolicy.sriovnetwork.openshift.io/policy-for-numa-0 created
```

2. **创建 SriovNetwork CR：**

a. **将以下 YAML 保存到 sriov-network.yaml 文件中，替换 YAML 中的值以匹配您的环境：**

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-numa-0-network (1)
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 (2)
  networkNamespace: <namespace> (3)
  ipam: |- (4)
    {
      "type": "<ipam_type>",
    }
```

(1) **将 sriov-numa-0-network 替换为 SR-IOV 网络资源的名称。**

(2) **指定上一步中的 SriovNetworkNodePolicy CR 的资源名称。此 YAML 使用示例 resourceName 值。**

(3) **输入 SR-IOV 网络资源的命名空间。**

(4) **输入 SR-IOV 网络的 IP 地址管理配置。**

b. **运行以下命令来创建 SriovNetwork 资源：**

```
$ oc create -f sriov-network.yaml
```

输出示例

```
sriovnetwork.sriovnetwork.openshift.io/sriov-numa-0-network created
```

3. **创建 pod 并从上一步中分配 SR-IOV 网络资源：**

a. **将以下 YAML 保存到 sriov-network-pod.yaml 文件中，替换 YAML 中的值以匹配您的环境：**

■

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod_name>
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "sriov-numa-0-network", (1)
        }
      ]
spec:
  containers:
  - name: <container_name>
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

1 这是使用 SriovNetworkNodePolicy 资源的 SriovNetwork 资源的名称。

b. 运行以下命令来创建 Pod 资源：

```
$ oc create -f sriov-network-pod.yaml
```

输出示例

```
pod/example-pod created
```

验证

1. 运行以下命令，将 <pod_name> 替换为 pod 的名称来验证 pod 的状态：

```
$ oc get pod <pod_name>
```

输出示例

```
NAME                                 READY   STATUS    RESTARTS   AGE
test-deployment-sriov-76cbbf4756-k9v72   1/1     Running   0          45h
```

2. 打开目标 pod 的 debug 会话，以验证 SR-IOV 网络资源是否已部署到与内存和 CPU 资源不同的节点上。

a. 运行以下命令，使用 pod 打开 debug 会话，将 <pod_name> 替换为目标 pod 名称。

```
$ oc debug pod/<pod_name>
```

b. 将 /host 设为 debug shell 中的根目录。debug pod 从 pod 中的/host 中的主机挂载 root 文件系统。将根目录改为 /host，您可以从主机文件系统中运行二进制文件：

```
$ chroot /host
```

c. 运行以下命令，查看有关 CPU 分配的信息：

```
$ lscpu | grep NUMA
```

输出示例

```
NUMA node(s):                2
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,...
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,...
```

```
$ cat /proc/self/status | grep Cpus
```

输出示例

```
Cpus_allowed: aa
Cpus_allowed_list: 1,3,5,7
```

```
$ cat  /sys/class/net/net1/device/numa_node
```

输出示例

```
0
```

在本例中，CPU 1,3,5 和 7 分配给 NUMA node1，但 SR-IOV 网络资源可以使用 NUMA node0 中的 NIC。

注意

如果 excludeTopology 规格被设置为 True，则同一 NUMA 节点上可能存在所需资源。

其他资源

- 使用 CPU Manager

25.4.6. 后续步骤

- 配置 SR-IOV 网络附加

25.5. 配置 SR-IOV 以太网网络附加

您可以为集群中的单根 I/O 虚拟化（SR-IOV）设备配置以太网网络附加。

25.5.1. 以太网设备配置对象

您可以通过定义 SriovNetwork 对象来配置以太网网络设备。

以下 YAML 描述了 SriovNetwork 对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
```

```
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: <sriov_resource_name> (3)
  networkNamespace: <target_namespace> (4)
  vlan: <vlan> (5)
  spoofChk: "<spoof_check>" (6)
  ipam: |- (7)
    {}
  linkState: <link_state> (8)
  maxTxRate: <max_tx_rate> (9)
  minTxRate: <min_tx_rate> (10)
  vlanQoS: <vlan_qos> (11)
  trust: "<trust_vf>" (12)
  capabilities: <capabilities> (13)
```

(1) 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。

(2) 安装 SR-IOV Network Operator 的命名空间。

(3) 用于为这个额外网络定义 SR-IOV 硬件的 SriovNetworkNodePolicy 对象中的 spec.resourceName 参数的值。

(4) SriovNetwork 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。

(5) 可选：额外网络的虚拟 LAN（VLAN）ID。它需要是一个从 0 到 4095 范围内的一个整数值。默认值为 0。

(6) 可选：VF 的 spoof 检查模式。允许的值是字符串 "on" 和 "off"。

重要

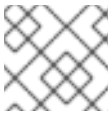
指定的值必须由引号包括，否则 SR-IOV Network Operator 将拒绝对象。

(7) 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

(8) 可选：虚拟功能（VF）的链接状态。允许的值是 enable、disable 和 auto。

(9) 可选：VF 的最大传输率（以 Mbps 为单位）。

(10) 可选：VF 的最低传输率（以 Mbps 为单位）。这个值必须小于或等于最大传输率。



注意

Intel NIC 不支持 minTxRate 参数。如需更多信息，请参阅 BZ#1772847。

(11) 可选：VF 的 IEEE 802.1p 优先级级别。默认值为 0。

(12) 可选：VF 的信任模式。允许的值是字符串 "on" 和 "off"。

重要

您必须在引号中包含指定的值，或者 SR-IOV Network Operator 拒绝对象。

13 **可选：为这个额外网络配置功能。您可以指定 '{ "ips": true }' 来启用 IP 地址支持，或指定 '{ "mac": true }' 来启用 MAC 地址支持。**

25.5.1.1. 为额外网络配置 IP 地址分配

IP 地址管理 (IPAM) Container Network Interface (CNI) 插件为其他 CNI 插件提供 IP 地址。

您可以使用以下 IP 地址分配类型：

- **静态分配。**
- **通过 DHCP 服务器进行动态分配。您指定的 DHCP 服务器必须可从额外网络访问。**
- **通过 Whereabouts IPAM CNI 插件进行动态分配。**

25.5.1.1.1. 静态 IP 地址分配配置

下表描述了静态 IP 地址分配的配置：

表 25.3. **ipam** 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。
Routes	**数组**	指定要在 pod 中配置的路由的一组对象。
dns	**数组**	可选：指定 DNS 配置的对象数组。

address 数组需要带有以下字段的对象：

表 25.4. **ipam.addresses[]** array

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如：如果您指定 **10.10.21.10/24**，那么会为额外网络分配 IP 地址 **10.10.21.10**，网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 25.5. **ipam.routes[]** array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	网络流量路由的网关。

表 25.6. **ipam.dns** object

字段	类型	描述
nameservers	**数组**	用于发送 DNS 查询的一个或多个 IP 地址的数组。
domain	**数组**	要附加到主机名的默认域。例如，如果将域设置为 **example.com**，对 **example-host** 的 DNS 查找查询将被改写为 **example-host.example.com**。
search	**数组**	在 DNS 查找查询过程中，附加到非限定主机名（如 **example-host**）的域名的数组。

静态 IP 地址分配配置示例

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7/24"
        }
      ]
  }
}
```

25.5.1.1.2. 动态 IP 地址(DHCP)分配配置

以下 JSON 描述了使用 DHCP 进行动态 IP 地址地址分配的配置。

DHCP 租期续订

pod 在创建时获取其原始 DHCP 租期。该租期必须由集群中运行的一个小型的 DHCP 服务器部署定期续订。

SR-IOV Network Operator 不创建 DHCP 服务器部署。Cluster Network Operator 负责创建小型的 DHCP 服务器部署。

要触发 DHCP 服务器的部署，您必须编辑 Cluster Network Operator 配置来创建 shim 网络附加，如下例所示：

shim 网络附加定义示例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
  # ...
```

表 25.7. ipam DHCP 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要值 dhcp。

动态 IP 地址(DHCP)分配配置示例

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

25.5.1.1.3. 使用 Whereabouts 进行动态 IP 地址分配配置

Whereabouts CNI 插件允许在不使用 DHCP 服务器的情况下动态地将 IP 地址分配给额外网络。

Whereabouts CNI 插件还支持在单独的 NetworkAttachmentDefinition 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

25.5.1.1.3.1. 动态 IP 地址配置对象

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 25.8. ipam whereabouts 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要 abouts 的值。
range	string	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	数组	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	string	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

25.5.1.1.3.2. 使用 Whereabouts 的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的动态地址分配配置：

Whereabouts 动态 IP 地址分配

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

25.5.1.1.3.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", (1)
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 2 的 network_name 匹配。

NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 1 的 network_name 匹配。

25.5.1.2. 为动态分配双栈 IP 地址创建配置

双栈 IP 地址分配可使用 ipRanges 参数进行配置：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 type 设置为 whereabouts。
2. 使用 ipRanges 来分配 IP 地址，如下例所示：

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. 将网络附加到 pod。如需更多信息，请参阅"将 pod 添加到额外网络"。
4. 验证是否分配了所有 IP 地址。

5. 运行以下命令，以确保 IP 地址被分配为元数据。

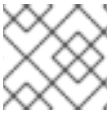
```
$ oc exec -it mypod -- ip a
```

其他资源

- 将 pod 附加到额外网络

25.5.2. 配置 SR-IOV 额外网络

您可以通过创建一个 SriovNetwork 对象来配置使用 SR-IOV 硬件的额外网络。创建SriovNetwork 对象时，SR-IOV Network Operator 会自动创建一个 NetworkAttachmentDefinition 对象。



注意

如果一个 SriovNetwork 对象已被附加到状态为 running 的 pod，则不要修改或删除它。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个 SriovNetwork 对象，然后在 <name>.yaml 文件中保存 YAML，其中 <name> 是这个额外网络的名称。对象规格可能类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. 运行以下命令来创建对象：

```
$ oc create -f <name>.yaml
```

这里的 <name> 指定额外网络的名称。

3. 可选：要确认与您在上一步中创建的 SriovNetwork 对象关联的 NetworkAttachmentDefinition 对象是否存在，请输入以下命令。将 <namespace> 替换为您在 SriovNetwork 对象中指定的 networkNamespace。

```
$ oc get net-attach-def -n <namespace>
```

25.5.3. 后续步骤

- 将 pod 添加到额外网络

25.5.4. 其他资源

- 配置 SR-IOV 网络设备

25.6. 配置 SR-IOV INFINIBAND 网络附加

您可以为集群中的单根 I/O 虚拟化（SR-IOV）设备配置 InfiniBand（IB）网络附加。

25.6.1. Infiniband 设备配置对象

您可以通过定义 SriovIBNetwork 对象来配置 InfiniBand（IB）网络设备。

以下 YAML 描述了 SriovIBNetwork 对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  ipam: |- 5
    {}
  linkState: <link_state> 6
  capabilities: <capabilities> 7
```

1. 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。
2. 安装 SR-IOV Operator 的命名空间。
3. 用于为这个额外网络定义 SR-IOV 硬件的 SriovNetworkNodePolicy 对象中的 spec.resourceName 参数的值。
4. SriovIBNetwork 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到网络设备。
5. 可选：将 IPAM CNI 插件配置为 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。
6. 可选：虚拟功能（VF）的链接状态。允许的值是 enable、disable 和 auto。
7. 可选：为此网络配置功能。您可以指定 '{ "ips": true }' 来启用 IP 地址支持，或 '{ "infinibandGUID": true }' 来启用 IB Global Unique Identifier (GUID)支持。

25.6.1.1. 为额外网络配置 IP 地址分配

IP 地址管理 (IPAM) Container Network Interface (CNI) 插件为其他 CNI 插件提供 IP 地址。

您可以使用以下 IP 地址分配类型：

- 静态分配。
- 通过 DHCP 服务器进行动态分配。您指定的 DHCP 服务器必须可从额外网络访问。
- 通过 Whereabouts IPAM CNI 插件进行动态分配。

25.6.1.1.1. 静态 IP 地址分配配置

下表描述了静态 IP 地址分配的配置：

表 25.9. ipam 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。
Routes	**数组**	指定要在 pod 中配置的路由的一组对象。
dns	**数组**	可选：指定 DNS 配置的对象数组。

address 数组需要带有以下字段的对象：

表 25.10. ipam.addresses[] array

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如：如果您指定 **10.10.21.10/24**，那么会为额外网络分配 IP 地址 **10.10.21.10**，网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 25.11. ipam.routes[] array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	网络流量路由的网关。

表 25.12. ipam.dns object

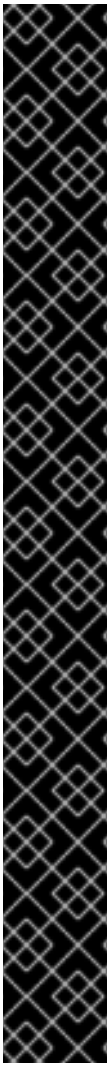
字段	类型	描述
nameservers	**数组**	用于发送 DNS 查询的一个或多个 IP 地址的数组。
domain	**数组**	要附加到主机名的默认域。例如，如果将域设置为 **example.com**，对 **example-host** 的 DNS 查找查询将被改写为 **example-host.example.com**。
search	**数组**	在 DNS 查找查询过程中，附加到非限定主机名（如 **example-host**）的域名的数组。

静态 IP 地址分配配置示例

```
{
  "ipam": {
    "type": "static",
     "addresses": [
       {
         "address": "191.168.1.7/24"
       }
     ]
  }
}
```

25.6.1.1.2. 动态 IP 地址(DHCP)分配配置

以下 JSON 描述了使用 DHCP 进行动态 IP 地址地址分配的配置。



DHCP 租期续订

pod 在创建时获取其原始 DHCP 租期。该租期必须由集群中运行的一个小型的 DHCP 服务器部署定期续订。

要触发 DHCP 服务器的部署，您必须编辑 Cluster Network Operator 配置来创建 shim 网络附加，如下例所示：

shim 网络附加定义示例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
  # ...
```

表 25.13. ipam DHCP 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要值 dhcp。

动态 IP 地址(DHCP)分配配置示例

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

25.6.1.1.3. 使用 Whereabouts 进行动态 IP 地址分配配置

Whereabouts CNI 插件允许在不使用 DHCP 服务器的情况下动态地将 IP 地址分配给额外网络。

Whereabouts CNI 插件还支持在单独的 NetworkAttachmentDefinition 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

25.6.1.1.3.1. 动态 IP 地址配置对象

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 25.14. ipam whereabouts 配置对象

字段	类型	描述
type	string	IPAM 地址类型。需要 abouts 的值。
range	string	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	数组	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	string	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

25.6.1.1.3.2. 使用 Whereabouts 的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的动态地址分配配置：

Whereabouts 动态 IP 地址分配

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

25.6.1.1.3.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", (1)
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 2 的 network_name 匹配。

NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 NetworkAttachmentDefinition 1 的 network_name 匹配。

25.6.1.2. 为动态分配双栈 IP 地址创建配置

双栈 IP 地址分配可使用 ipRanges 参数进行配置：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 type 设置为 whereabouts。
2. 使用 ipRanges 来分配 IP 地址，如下例所示：

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. 将网络附加到 pod。如需更多信息，请参阅"将 pod 添加到额外网络"。
4. 验证是否分配了所有 IP 地址。

5. 运行以下命令，以确保 IP 地址被分配为元数据。

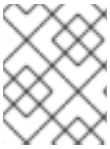
```
$ oc exec -it mypod -- ip a
```

其他资源

- 将 pod 附加到额外网络

25.6.2. 配置 SR-IOV 额外网络

您可以通过创建一个 SriovIBNetwork 对象来配置使用 SR-IOV 硬件的额外网络。创建SriovIBNetwork 对象时，SR-IOV Network Operator 会自动创建一个 NetworkAttachmentDefinition 对象。



注意

如果一个 SriovIBNetwork 对象已被附加到状态为 running 的 pod，则不要修改或删除它。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个 SriovIBNetwork 对象，然后在 <name>.yaml 文件中保存 YAML，其中 <name> 是这个额外网络的名称。对象规格可能类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. 运行以下命令来创建对象：

```
$ oc create -f <name>.yaml
```

这里的 <name> 指定额外网络的名称。

3. 可选：要确认与您在上一步中创建的 SriovIBNetwork 对象关联的 NetworkAttachmentDefinition 对象是否存在，请输入以下命令。将 <namespace> 替换为您在 SriovIBNetwork 对象中指定的 networkNamespace。

```
$ oc get net-attach-def -n <namespace>
```

25.6.3. 后续步骤

- 将 pod 添加到额外网络

25.6.4. 其他资源

- 配置 SR-IOV 网络设备

25.7. 将 POD 添加到额外网络

您可以将 pod 添加到现有的单根 I/O 虚拟化（SR-IOV）网络。

25.7.1. 网络附加的运行时配置

将 pod 附加到额外网络时，您可以指定运行时配置来为 pod 进行特定的自定义。例如,,您可以请求特定的 MAC 硬件地址。

您可以通过在 pod 规格中设置注解来指定运行时配置。注解键是 k8s.v1.cni.cncf.io/network,它接受一个 JSON 对象来描述运行时配置。

25.7.1.1. 基于以太网的 SR-IOV 附加的运行时配置

以下 JSON 描述了基于以太网的 SR-IOV 网络附加的运行时配置选项。

```
[
  {
    "name": "<name>", 1
    "mac": "<mac_address>", 2
    "ips": ["<cidr_range>"] 3
  }
]
```

1. SR-IOV 网络附加定义 CR 的名称。
2. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "mac": true }。
3. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "ips": true }。

运行时配置示例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
```

```
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

25.7.1.2. 基于 InfiniBand 的 SR-IOV 附加的运行时配置

以下 JSON 描述了基于 InfiniBand 的 SR-IOV 网络附加的运行时配置选项。

```
[
  {
    "name": "<network_attachment>", (1)
    "infiniband-guid": "<guid>", (2)
    "ips": ["<cidr_range>"] (3)
  }
]
```

(1) SR-IOV 网络附加定义 CR 的名称。

(2) SR-IOV 设备的 InfiniBand GUID。要使用这个功能，还必须在 SriovIBNetwork 对象中指定 { "infinibandGUID": true }。

(3) 从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能,你还必须在 SriovIBNetwork 对象中指定 { "ips": true }。

运行时配置示例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
```

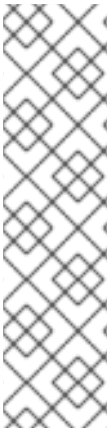
```
image: <image>
imagePullPolicy: IfNotPresent
command: ["sleep", "infinity"]
```

25.7.2. 将 pod 添加到额外网络

您可以将 pod 添加到额外网络。pod 继续通过默认网络发送与集群相关的普通网络流量。

创建 pod 时会附加额外网络。但是，如果 pod 已存在，您无法为其附加额外网络。

pod 必须与额外网络处于相同的命名空间。



注意

SR-IOV Network Resource Injector 会自动将 **resource** 字段添加到 pod 中的第一个容器中。

如果您在 Data Plane Development Kit(DPDK)模式下使用 Intel 网络接口控制器(NIC)，则只有 pod 中的第一个容器被配置为访问 NIC。如果在 **SriovNetworkNodePolicy** 对象中将 **deviceType** 设置为 **vfio-pci**，则您的 SR-IOV 额外网络被配置为 DPDK 模式。

您可以通过确保需要访问 NIC 的容器是 **Pod** 对象定义的第一个容器，或者禁用 Network Resource Injector(Network Resource Injector)来解决此问题。如需更多信息，请参阅 BZ#1990953。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 登录到集群。
- 安装 SR-IOV Operator。
- 创建 **SriovNetwork** 对象或 **SriovIBNetwork** 对象以将 pod 附加到。

流程

1. 为 **Pod** 对象添加注解。只能使用以下注解格式之一：

   a. 要在没有自定义的情况下附加额外网络，请使用以下格式添加注解。将 **<network>** 替换为要与 pod 关联的额外网络的名称：

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
   ```

   1 要指定多个额外网络，请使用逗号分隔各个网络。逗号之间不可包括空格。如果您多次指定同一额外网络，则该 pod 会将多个网络接口附加到该网络。

   b. 要通过自定义来附加额外网络，请添加具有以下格式的注解：

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: |-
   ```

```
[
  {
    "name": "<network>", 1
    "namespace": "<namespace>", 2
    "default-route": ["<default-route>"] 3
  }
]
```

1. 指定 NetworkAttachmentDefinition 对象定义的额外网络的名称。
2. 指定定义 NetworkAttachmentDefinition 对象的命名空间。
3. 可选：为默认路由指定覆盖，如 192.168.17.1。

2. 运行以下命令来创建 pod。将 <name> 替换为 pod 的名称。

```
$ oc create -f <name>.yaml
```

3. 可选：要确认 Pod CR 中是否存在注解，请输入以下命令将<name> 替换为 pod 的名称。

```
$ oc get pod <name> -o yaml
```

在以下示例中，example-pod pod 附加到net1 额外网络：

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

```
-
```

1. k8s.v1.cni.cncf.io/network-status 参数是对象的 JSON 数组。每个对象描述附加到 pod 的额外网络的状态。注解值保存为纯文本值。

25.7.3. 创建与 SR-IOV pod 兼容的非统一内存访问 (NUMA)

您可以通过限制 SR-IOV 和从相同 NUMA 节点分配的 CPU 资源，使用 **restricted** 或 **single-numa-node** Topology Manager 来创建与 SR-IOV pod 兼容的 NUMA。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您已将 CPU Manager 策略配置为 **static**。有关 CPU Manager 的详情请参考 "Additional resources" 部分。
- 您已将 Topology Manager 策略配置为 **single-numa-node**。

注意

当 **single-numa-node** 无法满足请求时，您可以将拓扑管理器策略配置为 **restricted**。有关更灵活的 SR-IOV 网络资源调度，请参阅*附加资源*部分中的 *NUMA 感知调度过程中排除 SR-IOV 网络拓扑*。

流程

1. 创建以下 SR-IOV pod 规格，然后在 **<name>-sriov-pod.yaml** 文件中保存 YAML。将 **<name>** 替换为这个 pod 的名称。
   以下示例显示了 SR-IOV pod 规格：

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> 1
spec:
  containers:
  - name: sample-container
    image: <image> 2
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" 3
        cpu: "2" 4
      requests:
        memory: "1Gi"
        cpu: "2"
```

1. 将 **<name>** 替换为 SR-IOV 网络附加定义 CR 的名称。
2. 将 **<image>** 替换为 **sample-pod** 镜像的名称。

3 要创建带有保证 QoS 的 SR-IOV pod，将 **memory limits** 设置为与 **memory requests** 相同的值。

4 要创建带有保证 QoS 的 SR-IOV pod，将 **cpu limits** 设置为与 **cpu requests** 相同。

2. 运行以下命令来创建 SR-IOV pod 示例：

```
$ oc create -f <filename> 1
```

1 将 <filename> 替换为您在上一步中创建的文件的名称。

3. 确认 **sample-pod** 配置为带有保证 QoS。

```
$ oc describe pod sample-pod
```

4. 确认 **sample-pod** 被分配了独有的 CPU。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. 确认为 **sample-pod** 分配的 SR-IOV 设备和 CPU 位于相同的 NUMA 节点上。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

25.7.4. 在 OpenStack 上使用 SR-IOV 的集群测试 pod 模板

以下 **testpmd** pod 演示了使用巨页、保留 CPU 和 SR-IOV 端口创建容器。

testpmd pod 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
```

```
      cpu: '2'
      openshift.io/sriov1: 1
    limits:
      hugepages-1Gi: 1Gi
      cpu: '2'
      memory: 1000Mi
      openshift.io/sriov1: 1
  volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
      readOnly: False
runtimeClassName: performance-cnf-performanceprofile (1)
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

1 本例假定性能配置集的名称为 cnf-performance profile。

25.7.5. 其他资源

- 配置 SR-IOV 以太网网络附加
- 配置 SR-IOV InfiniBand 网络附加
- 使用 CPU Manager
- 排除 NUMA 感知调度的 SR-IOV 网络拓扑

25.8. 为 SR-IOV 网络配置接口级网络 SYSCTL 设置和 ALL-MULTICAST 模式

作为集群管理员，您可以使用连接到 SR-IOV 网络设备的 pod 的 tuning Container Network Interface (CNI) meta 插件更改接口级网络 sysctl 和几个接口属性，如 promiscuous 模式、all-multicast 模式、MTU 和 MAC 地址。

25.8.1. 为启用了 SR-IOV 的 NIC 标记节点

如果您只想在 SR-IOV 功能的节点上启用 SR-IOV，请执行几种方法：

1. 安装 Node Feature Discovery (NFD) Operator。NFD 检测启用了 SR-IOV 的 NIC，并使用 **node.alpha.kubernetes-incubator.io/nfd-network-sriov.enabled = true** 标记节点。
2. 检查每个节点的 SriovNetworkNodeState CR。**interfaces** 小节包括 worker 节点上 SR-IOV Network Operator 发现的所有 SR-IOV 设备列表。使用以下命令，为每个节点标记 **feature.node.kubernetes.io/network-sriov.enabled: "true"** ：

```
$ oc label node <node_name> feature.node.kubernetes.io/network-
sriov.capable="true"
```

注意

您可以使用您需要的任何名称标记节点。

25.8.2. 设置一个 sysctl 标记

您可以为连接到 SR-IOV 网络设备的 pod 设置接口级网络 sysctl 设置。

在本例中，net.ipv4.conf.IFNAME.accept_redirects 在创建的虚拟接口上设置为 1。

sysctl-tuning-test 是本例中使用的命名空间。

- 使用以下命令来创建 sysctl-tuning-test 命名空间：

```
$ oc create namespace sysctl-tuning-test
```

25.8.2.1. 在使用 SR-IOV 网络设备的节点上设置一个 sysctl 标志

SR-IOV Network Operator 将 SriovNetworkNodePolicy.sriovnetwork.openshift.io 自定义资源定义 (CRD) 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。

注意

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空并重启节点。

它可能需要几分钟时间来应用配置更改。

按照以下步骤创建一个 SriovNetworkNodePolicy 自定义资源 (CR)。

流程

1. 创建一个 SriovNetworkNodePolicy 自定义资源 (CR)。例如，将以下 YAML 保存为文件 policyoneflag-sriov-node-network.yaml ：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyoneflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  nodeSelector: 4
    feature.node.kubernetes.io/network-sriov.capable="true"
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens5"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

1. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。

4 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。

5 可选： priority 是一个 0 到 99 之间的整数。较小的值具有更高的优先级。例如，优先级 10 是高于优先级 99。默认值为 99。

6 为 SR-IOV 物理网络设备创建的虚拟功能（VF）的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 128。

7 NIC 选择器标识要配置的 Operator 的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。如果指定了 rootDevices，则必须同时为 vendor、deviceID 或 pfNames 指定一个值。如果同时指定了 pfNames 和 rootDevices，请确保它们引用同一设备。如果您为 netFilter 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。

8 可选：该设备的一个或多个物理功能（PF）名称的数组。

9 可选：虚拟功能的驱动程序类型。唯一允许的值是 netdevice。对于裸机节点上的 DPDK 模式的 Mellanox NIC，请将 isRdma 设置为 true。

10 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 false。如果 isRdma 参数设为 true，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。将 isRdma 设置为 true，并将 needVhostNet 设置为 true 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。

注意

vfio-pci 驱动程序类型不被支持。

2. 创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

应用配置更新后，sriov-network-operator 命名空间中的所有 pod 将变为 Running 状态。

3. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 <node_name> 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

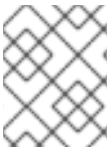
输出示例

```
Succeeded
```

25.8.2.2. 在 SR-IOV 网络中配置 sysctl

您可以通过将调优配置添加到 SriovNetwork 资源的可选 metaPlugins 参数，在 SR-IOV 创建的虚拟接口上设置特定于接口的 sysctl 设置。

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 NetworkAttachmentDefinition 自定义资源（CR）。



注意

不要编辑 SR-IOV Network Operator 所管理的 **NetworkAttachmentDefinition** 自定义资源。这样做可能会破坏额外网络上的网络流量。

要更改接口级别网络 **net.ipv4.conf.IFNAME.accept_redirects sysctl** 设置，请使用 Container Network Interface (CNI) 调整插件创建额外的 SR-IOV 网络。

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为额外 SR-IOV 网络附加创建 **SriovNetwork** 自定义资源 (CR) 并插入 **metaPlugins** 配置，如下例所示。将 YAML 保存为文件 **sriov-network-interface-sysctl.yaml**。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: onevalidflag (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: policyoneflag (3)
  networkNamespace: sysctl-tuning-test (4)
  ipam: '{ "type": "static" }' (5)
  capabilities: '{ "mac": true, "ips": true }' (6)
  metaPlugins : | (7)
   {
    "type": "tuning",
    "capabilities":{
      "mac":true
    },
    "sysctl":{
       "net.ipv4.conf.IFNAME.accept_redirects": "1"
    }
   }
```

(1) 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。

(2) 安装 SR-IOV Network Operator 的命名空间。

(3) 用于为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象中的 **spec.resourceName** 参数的值。

(4) **SriovNetwork** 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。

(5) 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

(6) 可选：为额外网络设置功能。您可以指定 "{ "ips": true }" 来启用 IP 地址支持，或指定 "{ "mac": true }" 来启用 MAC 地址支持。

7 可选： metaPlugins 参数用于为该设备添加额外的功能。在这种情况下，将 type 字段设置为 tuning。指定在 sysctl 字段中设置的接口级网络 sysctl。

2. 创建 SriovNetwork 资源：

```
$ oc create -f sriov-network-interface-sysctl.yaml
```

验证 NetworkAttachmentDefinition CR 是否已成功创建

- 运行以下命令，确认 SR-IOV Network Operator 创建了 NetworkAttachmentDefinition CR：

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 将 <namespace> 替换为您在 SriovNetwork 对象中指定的 networkNamespace 的值。例如： sysctl-tuning-test。

输出示例

```
NAME                 AGE
onevalidflag         14m
```

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

验证额外 SR-IOV 网络附加是否成功

要验证 tuning CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

1. 创建 Pod CR。将以下 YAML 保存为文件examplepod.yaml ：

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "onevalidflag", 1
          "mac": "0a:56:0a:83:04:0c", 2
          "ips": ["10.100.100.200/24"] 3
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
```

```
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1. SR-IOV 网络附加定义 CR 的名称。
2. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "mac": true }。
3. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "ips": true }。

2. 创建 Pod CR：

```
$ oc apply -f examplepod.yaml
```

3. 运行以下命令验证 pod 是否已创建：

```
$ oc get pod -n sysctl-tuning-test
```

输出示例

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

4. 运行以下命令登录到 pod：

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 验证配置的 sysctl 标记的值。运行以下命令，查找 net.ipv4.conf.IFNAME.accept_redirects 的值：

```
$ sysctl net.ipv4.conf.net1.accept_redirects
```

输出示例

```
net.ipv4.conf.net1.accept_redirects = 1
```

25.8.3. 为与绑定 SR-IOV 接口标记关联的 pod 配置 sysctl 设置

您可以为连接到绑定的 SR-IOV 网络设备的 pod 设置接口级网络 sysctl 设置。

在本例中，可以配置的特定网络接口级 sysctl 设置在绑定接口上设置。

sysctl-tuning-test 是本例中使用的命名空间。

- 使用以下命令来创建 sysctl-tuning-test 命名空间：

```
$ oc create namespace sysctl-tuning-test
```

25.8.3.1. 在带有绑定的 SR-IOV 网络设备的节点上设置所有 sysctl 标志

SR-IOV Network Operator 将 SriovNetworkNodePolicy.sriovnetwork.openshift.io 自定义资源定义 (CRD) 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。

注意

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。

它可能需要几分钟时间来应用配置更改。

按照以下步骤创建一个 SriovNetworkNodePolicy 自定义资源 (CR)。

流程

1. 创建一个 SriovNetworkNodePolicy 自定义资源 (CR)。将以下 YAML 保存为文件 policyallflags-sriov-node-network.yaml。将 policyallflags 替换为配置的名称。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyallflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
  nodeSelector: 4
    node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = `true`
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens1f0"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

1. 自定义资源对象的名称。
2. 安装 SR-IOV Network Operator 的命名空间。
3. SR-IOV 网络设备插件的资源名称。您可以为资源名称创建多个 SR-IOV 网络节点策略。
4. 节点选择器指定要配置的节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。
5. 可选： priority 是一个 0 到 99 之间的整数。较小的值具有更高的优先级。例如，优先级 10 是高于优先级 99。默认值为 99。
6.

为 SR-IOV 物理网络设备创建的虚拟功能 (VF) 的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能

7 NIC 选择器标识要配置的 Operator 的设备。您不必为所有参数指定值。建议您足够精确地识别网络设备以避免意外选择设备。如果指定了 rootDevices，则必须同时为 vendor、deviceID 或 pfNames 指定一个值。如果同时指定了 pfNames 和 rootDevices，请确保它们引用同一设备。如果您为 netFilter 指定了一个值，那么您不需要指定任何其他参数，因为网络 ID 是唯一的。

8 可选：该设备的一个或多个物理功能（PF）名称的数组。

9 可选：虚拟功能的驱动程序类型。唯一允许的值是 netdevice。对于裸机节点上的 DPDK 模式的 Mellanox NIC，请将 isRdma 设置为 true。

10 可选：配置是否启用远程直接访问 (RDMA) 模式。默认值为 false。如果 isRdma 参数设为 true，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。将 isRdma 设置为 true，并将 needVhostNet 设置为 true 以配置 Mellanox NIC 以用于 Fast Datapath DPDK 应用程序。

注意

vfio-pci 驱动程序类型不被支持。

2. 创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

应用配置更新后，sriov-network-operator 命名空间中的所有 pod 将变为 Running 状态。

3. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 <node_name> 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

输出示例

```
Succeeded
```

25.8.3.2. 在绑定的 SR-IOV 网络中配置 sysctl

您可以在从两个 SR-IOV 接口创建的绑定接口上设置特定于接口的 sysctl 设置。为此，可将调优配置添加到绑定网络附加定义的可选 Plugins 参数中。

注意

不要编辑 SR-IOV Network Operator 所管理的 NetworkAttachmentDefinition 自定义资源。这样做可能会破坏额外网络上的网络流量。

要更改特定的接口级网络 sysctl 设置，请按照以下流程使用 Container Network Interface (CNI) 调优插件创建 SriovNetwork 自定义资源 (CR)。

先决条件

先决条件

- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 为绑定接口创建 SriovNetwork 自定义资源 (CR)，如下例所示。将 YAML 保存为文件sriov-network-attachment.yaml。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: allvalidflags (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: policyallflags (3)
  networkNamespace: sysctl-tuning-test (4)
  capabilities: '{ "mac": true, "ips": true }' (5)
```

(1) 对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。

(2) 安装 SR-IOV Network Operator 的命名空间。

(3) 用于为这个额外网络定义 SR-IOV 硬件的 SriovNetworkNodePolicy 对象中的 spec.resourceName 参数的值。

(4) SriovNetwork 对象的目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络。

(5) 可选：为这个额外网络配置功能。您可以指定 "{ "ips": true }" 来启用 IP 地址支持，或指定 "{ "mac": true }" 来启用 MAC 地址支持。

2. 创建 SriovNetwork 资源：

```
$ oc create -f sriov-network-attachment.yaml
```

3. 创建绑定网络附加定义，如下例所示。将 YAML 保存为文件 sriov-bond-network-interface.yaml。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test
spec:
  config: '{
  "cniVersion":"0.4.0",
  "name":"bound-net",
  "plugins":[
   {
    "type":"bond", (1)
    "mode": "active-backup", (2)
```

```
    "failOverMac": 1, 3
    "linksInContainer": true, 4
    "miimon": "100",
    "links": [ 5
     {"name": "net1"},
     {"name": "net2"}
    ],
    "ipam":{ 6
     "type":"static"
    }
   },
   {
    "type":"tuning", 7
    "capabilities":{
     "mac":true
    },
    "sysctl":{
     "net.ipv4.conf.IFNAME.accept_redirects": "0",
     "net.ipv4.conf.IFNAME.accept_source_route": "0",
     "net.ipv4.conf.IFNAME.disable_policy": "1",
     "net.ipv4.conf.IFNAME.secure_redirects": "0",
     "net.ipv4.conf.IFNAME.send_redirects": "0",
     "net.ipv6.conf.IFNAME.accept_redirects": "0",
     "net.ipv6.conf.IFNAME.accept_source_route": "1",
     "net.ipv6.neigh.IFNAME.base_reachable_time_ms": "20000",
     "net.ipv6.neigh.IFNAME.retrans_time_ms": "2000"
    }
   }
  ]
}'
```

1. 类型是 bond。
2. mode 属性指定绑定模式。支持的绑定模式有：
   - balance-rr - 0
   - active-backup - 1
   - balance-xor - 2
     对于 balance-rr 或 balance-xor 模式，您必须为 SR-IOV 虚拟功能将 trust 模式设置为 on。
3. 对于 active-backup 模式，failover 属性是必需的。
4. linksInContainer=true 标志告知 Bond CNI 在容器内找到所需的接口。默认情况下，Bond CNI 会查找主机上的这些接口，该接口无法与 SRIOV 和 Multus 集成。
5. links 部分定义将用于创建绑定的接口。默认情况下，Multus 将附加的接口命名为 "net"，再加上一个连续的数字。
6. 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。在这个 pod 示例 IP 地址中被手动配置，因此在本例中 ipam 被设置为 static。
7. 为设备添加额外的功能。例如，将 type 字段设置为 tuning。指定在 sysctl 字段中设置的接口级网络 sysctl。这个示例设置可设置的所有接口级网络 sysctl 设置。

4. 创建绑定网络附加定义：

```
$ oc create -f sriov-bond-network-interface.yaml
```

验证 NetworkAttachmentDefinition CR 是否已成功创建

- 运行以下命令，确认 SR-IOV Network Operator 创建了 NetworkAttachmentDefinition CR：

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 将 <namespace> 替换为您在配置网络附加时指定的 networkNamespace，如 sysctl-tuning-test。

输出示例

```
NAME                 AGE
bond-sysctl-network  22m
allvalidflags        47m
```

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

验证额外的 SR-IOV 网络资源是否成功

要验证 tuning CNI 是否已正确配置并附加额外的 SR-IOV 网络附加，请执行以下操作：

1. 创建 Pod CR。例如，将以下 YAML 保存为文件 examplepod.yaml ：

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {"name": "allvalidflags"}, 1
        {"name": "allvalidflags"},
        {
          "name": "bond-sysctl-network",
          "interface": "bond0",
          "mac": "0a:56:0a:83:04:0c", 2
          "ips": ["10.100.100.200/24"] 3
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
```

```
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1. SR-IOV 网络附加定义 CR 的名称。
2. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "mac": true }。
3. 可选：从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "ips": true }。

2. 应用 YAML：

```
$ oc apply -f examplepod.yaml
```

3. 运行以下命令验证 pod 是否已创建：

```
$ oc get pod -n sysctl-tuning-test
```

输出示例

```
NAME     READY   STATUS    RESTARTS   AGE
tunepod  1/1     Running   0          47s
```

4. 运行以下命令登录到 pod：

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 验证配置的 sysctl 标记的值。运行以下命令，查找 net.ipv6.neigh.IFNAME.base_reachable_time_ms 的值：

```
$ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
```

输出示例

```
net.ipv6.neigh.bond0.base_reachable_time_ms = 20000
```

25.8.4. 关于 all-multicast 模式

启用 all-multicast 模式（特别是在无根应用程序上下文中）非常重要。如果没有启用此模式，则需要为 Pod 的安全上下文约束(SCC)授予 NET_ADMIN 功能。如果您要允许 NET_ADMIN 功能授予 pod 权限，以便更改超出其特定要求的更改，您可能会暴露安全漏洞。

tuning CNI 插件支持更改几个接口属性，包括 all-multicast 模式。通过启用此模式，您可以在 SR-IOV 网络设备上配置的虚拟功能 (VF) 上运行的应用程序从其他 VF 上的应用程序接收多播流量，无论是附加到同一还是不同的物理功能。

25.8.4.1. 在 SR-IOV 网络中启用 all-multicast 模式

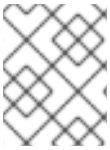
您可以通过以下方法在 SR-IOV 接口中启用 all-multicast 模式：

- 在 SriovNetwork 资源的 metaPlugins 参数中添加调优配置
- 在调优配置中将 allmulti 字段设置为 true

注意

确保创建启用了信任的虚拟功能 (VF)。

SR-IOV Network Operator 管理额外网络定义。当您指定要创建的额外 SR-IOV 网络时，SR-IOV Network Operator 会自动创建 NetworkAttachmentDefinition 自定义资源（CR）。



注意

不要编辑 SR-IOV Network Operator 所管理的 NetworkAttachmentDefinition 自定义资源。这样做可能会破坏额外网络上的网络流量。

按照本指南，在 SR-IOV 网络中启用 all-multicast 模式。

先决条件

- 已安装 OpenShift Container Platform CLI (oc)。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift Container Platform 集群。
- 已安装 SR-IOV Network Operator。
- 您已配置了适当的 SriovNetworkNodePolicy 对象。

流程

1. 使用以下设置创建一个 YAML 文件，为 Mellanox ConnectX-5 设备定义 SriovNetworkNodePolicy 对象。将 YAML 文件保存为 sriovnetpolicy-mlx.yaml。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-mlx
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    deviceID: "1017"
    pfNames:
      - ens8f0np0#0-9
    rootDevices:
      - 0000:d8:00.0
```

```
  vendor: "15b3"
nodeSelector:
  feature.node.kubernetes.io/network-sriov.capable: "true"
numVfs: 10
priority: 99
resourceName: resourcemlx
```

2. 可选：如果支持 SR-IOV 的集群节点还没有标记，请添加 SriovNetworkNodePolicy.Spec.NodeSelector 标签。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。

3. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f sriovnetpolicy-mlx.yaml
```

应用配置更新后，sriov-network-operator 命名空间中的所有 pod 会自动进入 Running 状态。

4. 运行以下命令来创建 enable-allmulti-test 命名空间：

```
$ oc create namespace enable-allmulti-test
```

5. 为额外 SR-IOV 网络附加创建 SriovNetwork 自定义资源 (CR) 并插入 metaPlugins 配置，如下例所示，并将该文件保存为 sriov-enable-all-multicast.yaml。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: enableallmulti (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: enableallmulti (3)
  networkNamespace: enable-allmulti-test (4)
  ipam: '{ "type": "static" }' (5)
  capabilities: '{ "mac": true, "ips": true }' (6)
  trust: "on" (7)
  metaPlugins : | (8)
   {
    "type": "tuning",
    "capabilities":{
     "mac":true
    },
    "allmulti": true
    }
   }
```

(1) 指定对象的名称。SR-IOV Network Operator 创建一个名称相同的 NetworkAttachmentDefinition 对象。

(2) 指定 SR-IOV Network Operator 安装到的命名空间。

(3) 指定来自用于为这个额外网络定义 SR-IOV 硬件的 SriovNetworkNodePolicy 对象的 spec.resourceName 参数的值。

(4) 为 SriovNetwork 对象指定目标命名空间。只有目标命名空间中的 pod 可以附加到额外网络

络。

5 为 IPAM CNI 插件指定一个配置对象作为 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

6 可选：为额外网络设置功能。您可以指定 "{ "ips": true }" 来启用 IP 地址支持，或指定 "{ "mac": true }" 来启用 MAC 地址支持。

7 指定虚拟功能的信任模式。这必须设置为 "on"。

8 使用 metaPlugins 参数为设备添加更多功能。在此用例中，将 type 字段设置为 tuning，并添加 allmulti 字段，并将它设为 true。

6. 运行以下命令来创建 SriovNetwork 资源：

```
$ oc create -f sriov-enable-all-multicast.yaml
```

验证 NetworkAttachmentDefinition CR

- 运行以下命令，确认 SR-IOV Network Operator 创建了 NetworkAttachmentDefinition CR：

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1 将 <namespace> 替换为您在 SriovNetwork 对象中指定的 networkNamespace 的值。在本例中，是 enable-allmulti-test。

输出示例

```
NAME                 AGE
enableallmulti         14m
```

注意

SR-IOV Network Operator 创建 CR 之前可能会有延迟。

1. 运行以下命令，显示 SR-IOV 网络资源的信息：

```
$ oc get sriovnetwork -n openshift-sriov-network-operator
```

验证额外的 SR-IOV 网络附加

要验证 tuning CNI 是否已正确配置并附加了额外的 SR-IOV 网络附加，请按照以下步骤执行：

1. 创建 Pod CR。将以下 YAML 示例保存到名为examplepod.yaml 的文件中：

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: enable-allmulti-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
```

```
      {
        "name": "enableallmulti", 1
        "mac": "0a:56:0a:83:04:0c", 2
        "ips": ["10.100.100.200/24"] 3
      }
     ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1 指定 SR-IOV 网络附加定义 CR 的名称。

2 可选：指定从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 MAC 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 {"mac": true}。

3 可选：指定从 SR-IOV 网络附加定义 CR 中定义的资源类型分配的 SR-IOV 设备的 IP 地址。支持 IPv4 和 IPv6 IP 地址。要使用这个功能，还必须在 SriovNetwork 对象中指定 { "ips": true }。

2. 运行以下命令来创建 Pod CR：

```
$ oc apply -f examplepod.yaml
```

3. 运行以下命令验证 pod 是否已创建：

```
$ oc get pod -n enable-allmulti-test
```

输出示例

```
NAME      READY   STATUS    RESTARTS   AGE
samplepod  1/1     Running   0          47s
```

4. 运行以下命令登录到 pod：

```
$ oc rsh -n enable-allmulti-test samplepod
```

5. 运行以下命令，列出与 pod 关联的所有接口：

```
sh-4.4# ip link
```

输出示例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue
state UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0 (1)
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0 (2)
```

1. eth0@if22 是主接口
2. net1@if24 是配置了 network-attachment-definition 的二级接口，它支持 all-multicast 模式(ALLMULTI 标志)

25.9. 为启用 SR-IOV 的工作负载配置 QINQ 支持

QinQ （正式称为 802.1Q-in-802.1Q）是由 IEEE 802.1ad 定义的联网技术。IEEE 802.1ad 扩展了 IEEE 802.1Q-1998 标准，并通过向已使用 802.1Q 标记的数据包引入额外的 802.1Q 标签增强 VLAN 功能。此方法也称为 VLAN 堆栈或双 VLAN。

25.9.1. 关于 802.1Q-in-802.1Q 支持

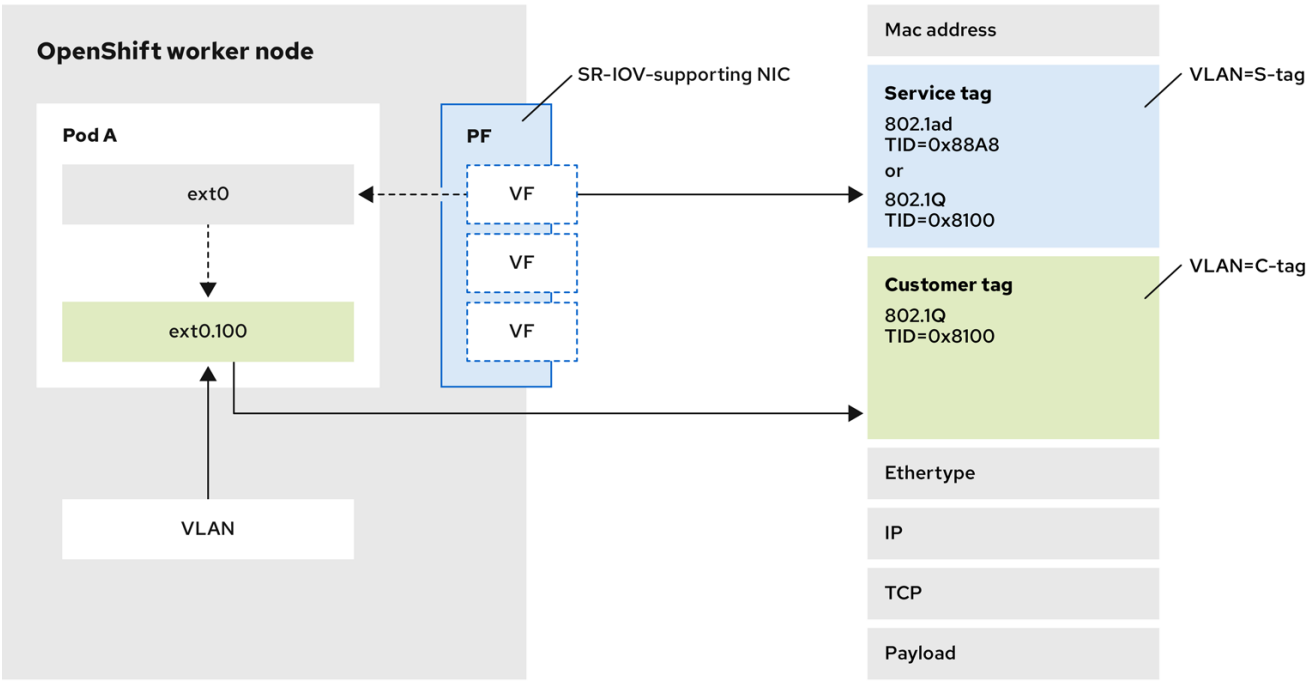
在传统的 VLAN 设置中，帧通常包含一个 VLAN 标签，如 VLAN-100，以及其他元数据，如服务质量 (QoS) 位和协议信息。QinQ 引入了第二个 VLAN 标签，其中服务提供商为其使用指定外部标签，提供它们的灵活性，而内部标签则保留给客户的 VLAN 专用。

QinQ 通过使用双 VLAN 标记促进嵌套 VLAN 的创建，从而在网络环境中实现更精细的分段和流量隔离。这种方法对于您需要通过通用基础架构向多个客户提供基于 VLAN 的服务的服务供应商网络特别有用，同时确保流量隔离和隔离。

下图演示了 OpenShift Container Platform 如何使用 SR-IOV 和 QinQ 实现容器化工作负载的高级网络分段和隔离。

图显示双 VLAN 标记 (QinQ) 如何在支持 SR-IOV 的 worker 节点上工作。位于 pod 命名空间中的 SR-IOV 虚拟功能 (VF)，ext0 由带有 VLAN ID 和 VLAN 协议的 SR-IOV Container Network Interface (CNI) 配置。这与 S-tag 对应。在 pod 中，VLAN CNI 使用主接口 ext0 创建一个子接口。此子接口使用 802.1Q 协议添加内部 VLAN ID，该协议对应于 C-tag。

这演示了 QinQ 如何在网络内启用更精细的流量分段和隔离。帧结构在右侧详细介绍，突出显示包含 VLAN 标签、EtherType、IP、TCP 和 Payload 部分的包含。QinQ 促进通过共享基础架构向多个客户提供基于 VLAN 的服务，同时确保流量隔离和隔离。



OpenShift Container Platform SR-IOV 解决方案支持在 SriovNetwork 自定义资源 (CR) 上设置 VLAN 协议。虚拟功能 (VF) 可以使用此协议来设置 VLAN 标签，也称为外部标签。然后，Pod 可以使用 VLAN CNI 插件来配置内部标签。

表 25.15. 支持的网络接口卡

NIC	802.1ad/802.1Q	802.1Q/802.1Q
Intel X710	否	支持
Intel E810	支持	支持
Mellanox	否	支持

其他资源

- 配置 VLAN 额外网络

25.9.2. 为启用 SR-IOV 的工作负载配置 QinQ 支持

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。

流程

1. 使用以下内容创建一个名为 sriovnetpolicy-810-sriov-node-network.yaml 的文件：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-810
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames:
      - ens5f0#0-9
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: resource810
```

2. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

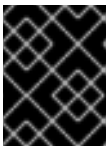
```
$ oc create -f sriovnetpolicy-810-sriov-node-network.yaml
```

3. 打开一个单独的终端窗口，运行以下命令来监控 openshift-sriov-network-operator 命名空间中指定节点的 SR-IOV 网络节点状态的同步状态：

```
$ watch -n 1 'oc get sriovnetworknodestates -n openshift-sriov-network-operator
<node_name> -o jsonpath="{.status.syncStatus}"'
```

同步状态表示从 InProgress 更改为 Succeeded。

4. 创建一个 SriovNetwork 对象，并设置名为 S-tag 或Service Tag 的外部 VLAN，因为它属于基础架构。



重要

您必须在交换机的中继接口上配置 VLAN。另外，您可能需要进一步配置一些交换机来支持 QinQ 标记。

a. 使用以下内容创建一个名为 nad-sriovnetwork-1ad-810.yaml 的文件：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriovnetwork-1ad-810
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{}'
  vlan: 171 (1)
  vlanProto: "802.1ad" (2)
  networkNamespace: default
  resourceName: resource810
```

(1) 将 S-tag VLAN 标签设置为 171。

2 指定要分配给虚拟功能 (VF) 的 VLAN 协议。支持的值是 802.1ad 和 802.1q。默认值为 802.1q。

b. 运行以下命令来创建对象：

```
$ oc create -f nad-sriovnetwork-1ad-810.yaml
```

5. 使用内部 VLAN 创建 NetworkAttachmentDefinition 对象。内部 VLAN 通常被称为 C-tag 或 Customer Tag，它属于 Network Function:

a. 使用以下内容，创建一个名为 nad-cvlan100.yaml 的文件：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: nad-cvlan100
  namespace: default
spec:
  config: '{
    "name": "vlan-100",
    "cniVersion": "0.3.1",
    "type": "vlan",
    "linkInContainer": true,
    "master": "net1", 1
    "vlanId": 100,
    "ipam": {"type": "static"}
  }'
```

1 指定 pod 中的 VF 接口。默认名称为 net1，因为 pod 注解中没有设置名称。

b. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f nad-cvlan100.yaml
```

验证

- 按照以下步骤，在节点上验证 QinQ 是否活跃：

1. 使用以下内容创建一个名为 test-qinq-pod.yaml 的文件：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: sriovnetwork-1ad-810, nad-cvlan100
spec:
  containers:
    - name: test-container
      image: quay.io/ocp-edge-qe/cnf-gotests-client:v4.10
      imagePullPolicy: Always
      securityContext:
        privileged: true
```

2. 运行以下命令来创建测试 pod：

```
$ oc create -f test-qinq-pod.yaml
```

3. 在存在 pod 的目标节点上进入 debug 会话，运行以下命令显示网络接口 ens5f0 的信息：

```
$ oc debug node/my-cluster-node -- bash -c "ip link show ens5f0"
```

输出示例

```
6: ens5f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state
UP mode DEFAULT group default qlen 1000
link/ether b4:96:91:a5:22:10 brd ff:ff:ff:ff:ff:ff
vf 0 link/ether a2:81:ba:d0:6f:f3 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 1 link/ether 8a:bb:0a:36:f2:ed brd ff:ff:ff:ff:ff:ff, vlan 171, vlan protocol 802.1ad,
spoof checking on, link-state auto, trust off
vf 2 link/ether ca:0e:e1:5b:0c:d2 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 3 link/ether ee:6c:e2:f5:2c:70 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 4 link/ether 0a:d6:b7:66:5e:e8 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 5 link/ether da:d5:e7:14:4f:aa brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 6 link/ether d6:8e:85:75:12:5c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 7 link/ether d6:eb:ce:9c:ea:78 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
vf 8 link/ether 5e:c5:cc:05:93:3c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust on
vf 9 link/ether a6:5a:7c:1c:2a:16 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state
auto, trust off
```

输出中的 vlan protocol 802.1ad ID 表示接口支持带有协议 802.1ad (QinQ)的 VLAN 标记。VLAN ID 为 171。

25.10. 配置高性能多播

您可以在您的单根 I/O 虚拟化（SR-IOV）硬件网络中使用多播。

25.10.1. 高性能多播

OpenShift SDN 网络插件支持默认网络上的 pod 间的多播。目前，多播最适用于低带宽协调或服务发现。它不适用于高带宽的应用程序。对于流传输介质应用程序，如 IPTV 和多方视频会议，可以使用 Single Root I/O Virtualization（SR-IOV）硬件来提供接近原生的性能。

使用额外的 SR-IOV 接口进行多播时：

- pod 必须通过额外的 SR-IOV 接口发送或接收多播软件包。
- 连接 SR-IOV 接口的物理网络决定了多播路由和拓扑结构，不受 OpenShift Container Platform 的控制。

25.10.2. 为多播配置 SR-IOV 接口

以下步骤为多播创建一个 SR-IOV 接口示例。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

1. 创建一个 SriovNetworkNodePolicy 对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. 创建一个 SriovNetwork 对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | (1)
    {
      "type": "host-local", (2)
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

(1) (2) 如果选择将 DHCP 配置为 IPAM，请确保通过 DHCP 服务器提供了以下默认路由：224.0.0.0/5 和 232.0.0.0/5。这会覆盖由默认网络供应商设置的静态多播路由。

3. 创建带有多播应用程序的 pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] 1
    command: [ "sleep", "infinity"]
```

1 只有在应用程序需要为 SR-IOV 接口分配多播 IP 地址时，才需要 NET_ADMIN 功能。否则，可以省略它。

25.11. 使用 DPDK 和 RDMA

OpenShift Container Platform 支持容器化 Data Plane Development Kit (DPDK) 应用程序。您可以使用单一根 I/O 虚拟化（SR-IOV）网络硬件和 Data Plane Development Kit (DPDK) 以及远程直接内存访问 (RDMA) 。

有关支持的设备的详情，请参考支持的设备。

25.11.1. 在 DPDK 模式中使用 Intel NIC 的虚拟功能

先决条件

- 安装 OpenShift CLI（oc）。
- 安装 SR-IOV Network Operator。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建以下 SriovNetworkNodePolicy 对象，然后在 intel-dpdk-node-policy.yaml 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

```
    priority: <priority>
    numVfs: <num>
    nicSelector:
      vendor: "8086"
      deviceID: "158b"
      pfNames: ["<pf_name>", ...]
      rootDevices: ["<pci_bus_id>", "..."]
    deviceType: vfio-pci (1)
```

1 将虚拟功能（VF）的驱动器类型指定为 vfio-pci。

注意

如需了解 inSriovNetworkNodePolicy 的每个选项的详情，请参阅 Configuring SR-IOV network devices 部分。

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，openshift-sriov-network-operator 命名空间中的所有 pod 将变为 Running 状态。

2. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f intel-dpdk-node-policy.yaml
```

3. 创建以下 SriovNetwork 对象，然后在 intel-dpdk-network.yaml 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |-
# ... (1)
  vlan: <vlan>
  resourceName: intelnics
```

1 为 ipam CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

注意

如需 SriovNetwork 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 app-netutil 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 SriovNetwork 对象：

```
$ oc create -f intel-dpdk-network.yaml
```

5. 创建以下 Pod spec，然后在 intel-dpdk-pod.yaml 文件中保存 YAML。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. 指定 target_namespace，它与 SriovNetwork 对象 intel-dpdk-network 创建于的命令空间相同。如果要在其他命名空间中创建 pod，在 Pod spec 和 SriovNetwork 对象中更改 target_namespace。

2. 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。

3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。

4. 在 /mnt/huge 下将巨页卷挂载到 DPDK pod。巨页卷由 emptyDir 卷类型支持，medium 为Hugepages。

5. 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果未明确指定，则此资源请求和限制将被 SR-IOV 网络资源注入程序自动添加。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 SriovOperatorConfig CR 中的 enableInjector 选项设置为 false 来禁用它。

6 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 static，并创建带有有保障的 QoS 的 pod 来实现。

7 指定巨页大小 hugepages-1Gi 或 hugepages-2Mi 以及分配给 DPDK pod 的巨页数量。单独配置 2Mi 和 1Gi 巨页。配置 1Gi 巨页需要在节点中添加内核参数。例如：添加内核参数 default_hugepagesz=1GB，hugepagesz=1G 和 hugepages=16 将在系统引导时分配 16*1Gi 巨页。

6. 运行以下命令来创建 DPDK pod:

```
$ oc create -f intel-dpdk-pod.yaml
```

25.11.2. 在带有 Mellanox NIC 的 DPDK 模式中使用虚拟功能

您可以创建一个网络节点策略，并在带有 Mellanox NIC 的 DPDK 模式中使用虚拟功能创建 Data Plane Development Kit (DPDK) pod。

先决条件

- 已安装 OpenShift CLI（oc）。
- 已安装 Single Root I/O Virtualization (SR-IOV) Network Operator。
- 您已以具有 cluster-admin 权限的用户身份登录。

流程

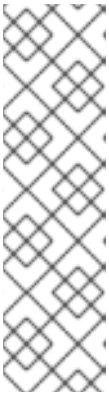
1. 将以下 SriovNetworkNodePolicy YAML 配置保存到 mlx-dpdk-node-policy.yaml 文件中：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

1 指定 SR-IOV 网络设备的设备十六进制代码。

2 指定到 netdevice 的虚拟功能（VF）的驱动器类型。Mellanox SR-IOV 虚拟功能 (VF) 可以在 DPDK 模式下工作，而无需使用 vfio-pci 设备类型。VF 设备作为容器内的内核网络接口显示。

3 启用远程直接内存访问 (RDMA) 模式。Mellanox 卡需要在 DPDK 模式下工作。



注意

如需了解 SriovNetworkNodePolicy 对象中的每个选项的详细说明，请参阅*配置 SR-IOV 网络设备*。

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，openshift-sriov-network-operator 命名空间中的所有 pod 将变为 Running 状态。

2. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

3. 将以下 SriovNetwork YAML 配置保存到 mlx-dpdk-network.yaml 文件中：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- 1
...
  vlan: <vlan>
  resourceName: mlxnics
```

1 为 IP 地址管理 (IPAM) Container Network Interface (CNI) 插件指定一个配置对象作为 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

注意

如需了解 SriovNetwork 对象中的每个选项的详细说明，请参阅*配置 SR-IOV 网络设备*。

app-netutil 选项库提供了几个 API 方法，用于收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 SriovNetwork 对象：

```
$ oc create -f mlx-dpdk-network.yaml
```

5. 将以下 Pod YAML 配置保存到 mlx-dpdk-pod.yaml 文件中：

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. 指定 target_namespace，它与 SriovNetwork 对象 mlx-dpdk-network 创建于的命令空间相同。要在不同命名空间中创建 pod，在 Pod spec 和 SriovNetwork 对象中更改 target_namespace。
2. 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。
3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。
4. 将巨页卷挂载到 /mnt/huge 下的 DPDK pod 中。巨页卷由 emptyDir 卷类型支持，介质是 Hugepages。
5. 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果没有明确指定，则 SR-IOV 网络资源注入程序会自动添加此资源请求和限制。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 SriovOperatorConfig CR 中的 enableInjector 选项设置为 false 来禁用它。
6. 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。要做到这一点，将 CPU Manager 策略设置为 static，并创建带有 Guaranteed 服务质量 (QoS) 的 pod。
7. 指定巨页大小 hugepages-1Gi 或 hugepages-2Mi 以及分配给 DPDK pod 的巨页数量。单独配置 2Mi 和 1Gi 巨页。配置 1Gi 巨页需要在节点中添加内核参数。

6. 运行以下命令来创建 DPDK pod:

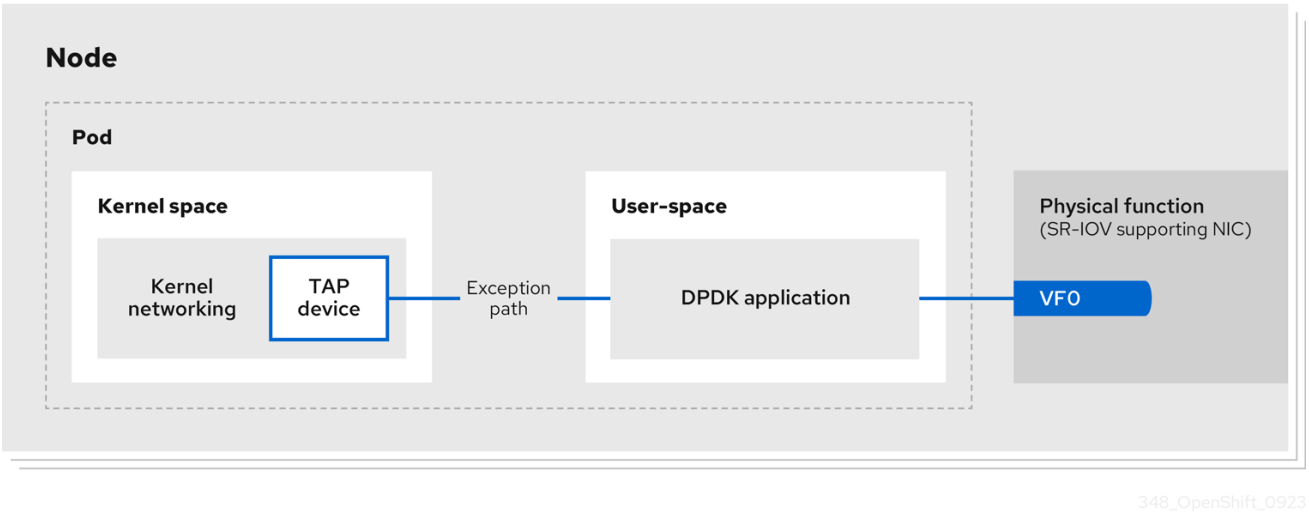
```
$ oc create -f mlx-dpdk-pod.yaml
```

25.11.3. 使用TAP CNI 运行具有内核访问权限的 rootless DPDK 工作负载

DPDK 应用程序可以使用 virtio-user 作为异常路径，将某些类型的数据包（如日志消息）注入内核进行处理。有关此功能的更多信息，请参阅 Virtio_user 作为例外路径。

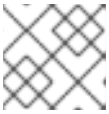
在 OpenShift Container Platform 版本 4.14 及更高版本中，您可以使用非特权 pod 和 tap CNI 插件运行 DPDK 应用程序。要启用此功能，您需要在 SriovNetworkNodePolicy 对象中将 needVhostNet 参数设置为 true 来挂载 vhost-net 设备。

图 25.1. DPDK 和TAP 示例配置



先决条件

- 已安装 OpenShift CLI（oc）。
- 已安装 SR-IOV Network Operator。
- 您以具有 cluster-admin 权限的用户身份登录。
- 确保在所有节点上将 container_use_devices=on 设置为 root。



注意

使用 Machine Config Operator 设置此 SELinux 布尔值。

流程

1. 创建一个文件，如 test-namespace.yaml，其内容类似以下示例：

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
```

```
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
```

2. 运行以下命令来创建新的 Namespace 对象：

```
$ oc apply -f test-namespace.yaml
```

3. 创建一个文件，如 sriov-node-network-policy.yaml，内容类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice (1)
 isRdma: true (2)
 needVhostNet: true (3)
 nicSelector:
   vendor: "15b3" (4)
   deviceID: "101b" (5)
   rootDevices: ["00:05.0"]
 numVfs: 10
 priority: 99
 resourceName: sriovnic
 nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

(1) 这表示配置集是为 Mellanox Network Interface Controller (NIC) 量身定制的。

(2) 只有 Mellanox NIC 才需要将 isRdma 设置为 true。

(3) 这会将 /dev/net/tun 和 /dev/vhost-net 设备挂载到容器中，以便应用可以创建 tap 设备，并将 tap 设备连接到 DPDK 工作负载。

(4) SR-IOV 网络设备厂商的十六进制代码。值 15b3 代表与 Mellanox NIC 关联。

(5) SR-IOV 网络设备的设备十六进制代码。

4. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f sriov-node-network-policy.yaml
```

5. 创建以下 SriovNetwork 对象，然后在 sriov-network-attachment.yaml 文件中保存 YAML：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
```

```
resourceName: sriovnic
spoofChk: "off"
trust: "on"
```

注意

如需 SriovNetwork 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 app-netutil 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

6. 运行以下命令来创建 SriovNetwork 对象：

```
$ oc create -f sriov-network-attachment.yaml
```

7. 创建一个文件，如 tap-example.yaml，该文件定义网络附加定义，其内容类似以下示例：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: tap-one
 namespace: test-namespace (1)
spec:
 config: '{
   "cniVersion": "0.4.0",
   "name": "tap",
   "plugins": [
     {
       "type": "tap",
       "multiQueue": true,
       "selinuxcontext": "system_u:system_r:container_t:s0"
     },
     {
       "type":"tuning",
       "capabilities":{
         "mac":true
       }
     }
   ]
 }'
```

(1) 指定创建 SriovNetwork 对象的 target_namespace。

8. 运行以下命令来创建 NetworkAttachmentDefinition 对象：

```
$ oc apply -f tap-example.yaml
```

9. 创建一个文件，如 dpdk-pod-rootless.yaml，其内容类似以下示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
```

```
  namespace: test-namespace (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {"name": "sriov-network", "namespace": "test-namespace"},
      {"name": "tap-one", "interface": "ext0", "namespace": "test-namespace"}]'
spec:
  nodeSelector:
    kubernetes.io/hostname: "worker-0"
  securityContext:
      fsGroup: 1001 (2)
      runAsGroup: 1001 (3)
      seccompProfile:
        type: RuntimeDefault
  containers:
  - name: testpmd
    image: <DPDK_image> (4)
    securityContext:
      capabilities:
        drop: ["ALL"] (5)
        add: (6)
          - IPC_LOCK
          - NET_RAW #for mlx only (7)
      runAsUser: 1001 (8)
      privileged: false (9)
      allowPrivilegeEscalation: true (10)
      runAsNonRoot: true (11)
    volumeMounts:
    - mountPath: /mnt/huge (12)
      name: hugepages
    resources:
      limits:
        openshift.io/sriovnic: "1" (13)
        memory: "1Gi"
        cpu: "4" (14)
        hugepages-1Gi: "4Gi" (15)
      requests:
        openshift.io/sriovnic: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  runtimeClassName: performance-cnf-performanceprofile (16)
  volumes:
  - name: hugepages
    emptyDir:
      medium: HugePages
```

(1) 指定 target_namespace 创建 SriovNetwork 对象。如果要在其他命名空间中创建 pod，在 Pod spec 和 SriovNetwork 对象中更改 target_namespace。

(2) 设置在这些卷中创建的卷挂载目录和文件的组所有权。

(3) 指定用于运行容器的主要组 ID。

4 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。

5 从容器的 securityContext 中删除所有功能 (ALL) 意味着容器在正常操作之外没有特殊的特权。

6 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。这些功能还必须使用 setcap 命令在二进制文件中设置。

7 Mellanox 网络接口控制器(NIC)需要 NET_RAW 功能。

8 指定用于运行容器的用户 ID。

9 此设置表示 pod 中的容器或容器不应被授予对主机系统的特权访问权限。

10 此设置允许容器在可能已分配的初始非 root 特权外升级其特权。

11 此设置可确保容器以非 root 用户身份运行。这有助于强制实施最小特权的原则，限制对容器造成潜在的影响，并减少攻击面。

12 在 /mnt/huge 下将巨页卷挂载到 DPDK pod。巨页卷由 emptyDir 卷类型支持，medium 为Hugepages。

13 可选：指定分配给 DPDK pod 的 DPDK 设备数。如果没有明确指定，则 SR-IOV 网络资源注入程序会自动添加此资源请求和限制。SR-IOV 网络资源注入程序是由 SR-IOV Operator 管理的准入控制器组件。它默认是启用的，可以通过把默认的 SriovOperatorConfig CR 中的 enableInjector 选项设置为 false 来禁用它。

14 指定 CPU 数量。DPDK pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 static，并创建带有有保障的 QoS 的 pod 来实现。

15 指定巨页大小 hugepages-1Gi 或 hugepages-2Mi 以及分配给 DPDK pod 的巨页数量。单独配置 2Mi 和 1Gi 巨页。配置 1Gi 巨页需要在节点中添加内核参数。例如：添加内核参数 default_hugepagesz=1GB，hugepagesz=1G 和 hugepages=16 将在系统引导时分配 16*1Gi 巨页。

16 如果您的性能配置集没有命名为 cnf-performance profile，请将该字符串替换为正确的性能配置集名称。

10. 运行以下命令来创建 DPDK pod:

```
$ oc create -f dpdk-pod-rootless.yaml
```

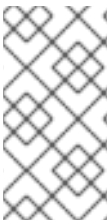
其他资源

- 启用 container_use_devices 布尔值
- 创建性能配置集
- 配置 SR-IOV 网络设备

25.11.4. 实现特定 DPDK 行率概述

要实现特定的 Data Plane Development Kit (DPDK) 行率，部署 Node Tuning Operator 并配置单根 I/O 虚拟化 (SR-IOV)。您还必须为以下资源调整 DPDK 设置：

- 隔离的 CPU
- Hugepages
- 拓扑调度程序

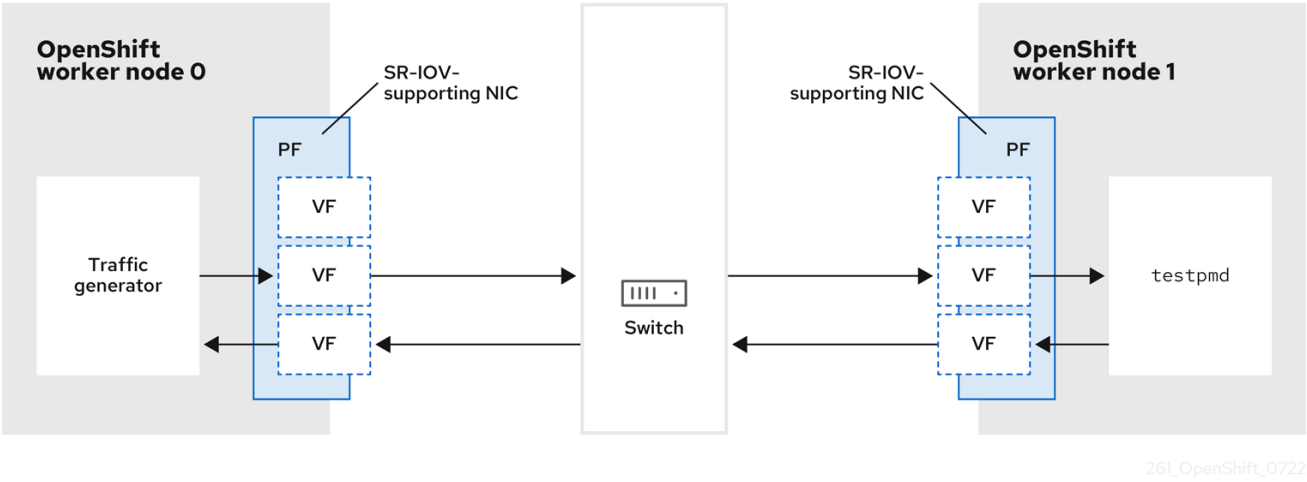


注意

在早期版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift Container Platform 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

DPDK 测试环境

下图显示了流量测试环境的组件：



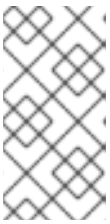
- 流量生成器：可以生成高容量数据包流量的应用。
- SR-IOV 支持 NIC：与 SR-IOV 兼容的网络接口卡。该卡在物理接口上运行多个虚拟功能。
- 物理功能 (PF)：支持 SR-IOV 接口的网络适配器的 PCI Express (PCIe) 功能。
- 虚拟功能 (VF)：支持 SR-IOV 的网络适配器上的轻量级 PCIe 功能。VF 与网络适配器上的 PCIe PF 关联。VF 代表网络适配器的虚拟实例。
- 交换机：网络交换机。节点也可以重新连接到回来。
- testpmd：DPDK 中包含的示例应用程序。testpmd 应用可用于在数据包转发模式下测试 DPDK。testpmd 应用程序也是如何使用 DPDK 软件开发套件 (SDK) 构建功能全面的应用程序的示例。
- worker 0 和 worker 1：OpenShift Container Platform 节点。

25.11.5. 使用 SR-IOV 和 Node Tuning Operator 实现 DPDK 行率

您可以使用 Node Tuning Operator 配置隔离的 CPU、巨页和拓扑调度程序。然后，您可以使用 Node Tuning Operator 和单根 I/O 虚拟化 (SR-IOV) 来实现特定的 Data Plane Development Kit (DPDK) 行率。

先决条件

- 已安装 OpenShift CLI（oc）。
- 已安装 SR-IOV Network Operator。
- 您已以具有 cluster-admin 权限的用户身份登录。
- 您已部署了独立的 Node Tuning Operator。



注意

在以前版本的 OpenShift Container Platform 中，Performance Addon Operator 用来实现自动性能优化，以便为 OpenShift 应用程序实现低延迟性能。在 OpenShift Container Platform 4.11 及更新的版本中，这个功能是 Node Tuning Operator 的一部分。

流程

1. 根据以下示例创建 PerformanceProfile 对象：

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  globallyDisableIrqLoadBalancing: true
  cpu:
    isolated: 21-51,73-103 (1)
    reserved: 0-20,52-72 (2)
  hugepages:
    defaultHugepagesSize: 1G (3)
    pages:
      - count: 32
        size: 1G
  net:
    userLevelNetworking: true
  numa:
    topologyPolicy: "single-numa-node"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

(1) 如果系统上启用了超线程，请分配与 isolated 和 reserved CPU 组的相关符号链接。如果系统包含多个非统一内存访问节点 (NUMA)，请将两个 NUMA 中的 CPU 分配给这两个组。您还可以将 Performance Profile Creator 用于此任务。如需更多信息，请参阅*创建性能配置集*。

(2) 您还可以指定将队列设置为保留 CPU 数的设备列表。如需更多信息，请参阅*使用 Node Tuning Operator 缩减 NIC 队列*。

(3) 分配所需的巨页的数量和大小。您可以为巨页指定 NUMA 配置。默认情况下，系统会为系统上的每个 NUMA 节点分配一个偶数数量。如果需要，您可以请求对节点使用实时内核。如需更多信息，请参阅*置备具有实时功能的 worker*。

2. 将 yaml 文件保存为 mlx-dpdk-perfprofile-policy.yaml。

3. 使用以下命令应用性能配置集：

```
$ oc create -f mlx-dpdk-perfprofile-policy.yaml
```

25.11.5.1. 用于虚拟功能的 SR-IOV Network Operator 示例

您可以使用单根 I/O 虚拟化 (SR-IOV) Network Operator 从节点上分配和配置虚拟功能 (VF) 的虚拟功能 (VF)。

如需有关部署 Operator 的更多信息，请参阅 *安装 SR-IOV Network Operator*。有关配置 SR-IOV 网络设备的更多信息，请参阅 *配置 SR-IOV 网络设备*。

在 Intel VF 和 Mellanox VF 上运行 Data Plane Development Kit (DPDK) 工作负载之间存在一些区别。本节为这两个 VF 类型提供对象配置示例。以下是用于在 Intel NIC 上运行 DPDK 应用程序的 **sriovNetworkNodePolicy** 对象示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci 1
  needVhostNet: true 2
  nicSelector:
    pfNames: ["ens3f0"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  needVhostNet: true
  nicSelector:
    pfNames: ["ens3f1"]
  nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_2
```

1. 对于 Intel NIC，**deviceType** 必须是 **vfio-pci**。
2. 如果需要内核与 DPDK 工作负载通信，请添加 **needVhostNet: true**。这会将 **/dev/net/tun** 和 **/dev/vhost-net** 设备挂载到容器中，以便应用可以创建 tap 设备，并将 tap 设备连接到 DPDK 工作负载。

以下是 Mellanox NIC 的 **sriovNetworkNodePolicy** 对象示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice (1)
  isRdma: true (2)
  nicSelector:
    rootDevices:
      - "0000:5e:00.1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-2
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    rootDevices:
      - "0000:5e:00.0"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_2
```

(1) 对于 Mellanox 设备，**deviceType** 必须是 **netdevice**。

(2) 对于 Mellanox 设备，**isRdma** 必须为 **true**。Mellanox 卡使用流 Bifurcation 连接到 DPDK 应用程序。这种机制在 Linux 用户空间和内核空间之间分割流量，并增强了行速率处理能力。

25.11.5.2. SR-IOV 网络 Operator 示例

以下是 **sriovNetwork** 对象的示例定义。在这种情况下，Intel 和 Mellanox 配置是相同的：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-1
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.1.0/24"}]],"dataDir":
    "/run/my-orchestrator/container-ipam-state-1"}' (1)
  networkNamespace: dpdk-test (2)
```

```
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_1 3
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-2
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.2.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'
  networkNamespace: dpdk-test
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_2
```

1 您可以使用不同的 IP 地址管理 (IPAM) 实现，如 Whereabouts。如需更多信息，请参阅*使用 Whereabouts 进行动态 IP 地址分配配置*。

2 您必须请求创建网络附加定义的 networkNamespace。您必须在 openshift-sriov-network-operator 命名空间内创建 sriovNetwork CR。

3 resourceName 值必须与在 sriovNetworkNodePolicy 下创建的 resourceName 相匹配。

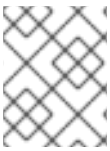
25.11.5.3. DPDK 基础工作负载示例

以下是数据平面开发套件 (DPDK) 容器的示例：

```
apiVersion: v1
kind: Namespace
metadata:
  name: dpdk-test
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[ 1
     {
      "name": "dpdk-network-1",
      "namespace": "dpdk-test"
     },
     {
      "name": "dpdk-network-2",
      "namespace": "dpdk-test"
     }
    ]'
    irq-load-balancing.crio.io: "disable" 2
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  labels:
    app: dpdk
  name: testpmd
```

```
  namespace: dpdk-test
spec:
  runtimeClassName: performance-performance
  containers:
    - command:
        - /bin/bash
        - -c
        - sleep INF
      image: registry.redhat.io/openshift4/dpdk-base-rhel8
      imagePullPolicy: Always
      name: dpdk
      resources:
        limits:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
        requests:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
      securityContext:
        capabilities:
          add:
            - IPC_LOCK
            - SYS_RESOURCE
            - NET_RAW
            - NET_ADMIN
        runAsUser: 0
      volumeMounts:
        - mountPath: /mnt/huge
          name: hugepages
  terminationGracePeriodSeconds: 5
  volumes:
    - emptyDir:
        medium: HugePages
      name: hugepages
```

1. 请求您需要的 SR-IOV 网络。设备的资源将自动注入。
2. 禁用 CPU 和 IRQ 负载均衡基础。如需更多信息，请参阅 *禁用单个 pod 的中断处理*。
3. 将 runtimeClass 设置为 performance-performance。不要将 runtimeClass 设置为 HostNetwork 或 privileged。
4. 为请求和限值请求相同数量的资源，以启动具有 Guaranteed 服务质量 (QoS) 的 pod。



注意

不要使用 SLEEP 启动容器集，然后执行到容器集以启动 testpmd 或 DPDK 工作负载。这可添加额外的中断，因为 exec 进程没有固定到任何 CPU。

25.11.5.4. testpmd 脚本示例

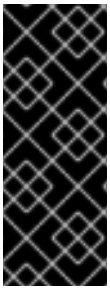
以下是运行 testpmd 的示例脚本：

```
#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -l ${CPU} -a ${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxd=4096 --txd=4096 --
rxq=7 --txq=7 --forward-mode=mac --eth-peer=0,50:00:00:00:00:01 --eth-
peer=1,50:00:00:00:00:02
```

这个示例使用两个不同的 sriovNetwork CR。环境变量包含分配给 pod 的虚拟功能 (VF) PCI 地址。如果您在 pod 定义中使用相同的网络，您必须分割 pciAddress。配置流量生成器的正确 MAC 地址非常重要。这个示例使用自定义 MAC 地址。

25.11.6. 在带有 Mellanox NIC 的 RDMA 模式中使用虚拟功能



重要

RDMA over Converged Ethernet (RoCE) 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

在 OpenShift Container Platform 上使用 RDMA 时，RDMA over Converged Ethernet (RoCE) 是唯一支持的模式。

先决条件

- 安装 OpenShift CLI（oc）。
- 安装 SR-IOV Network Operator。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建以下 SriovNetworkNodePolicy 对象，然后在 mlx-rdma-node-policy.yaml 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
```

```
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) 指定 SR-IOV 网络设备的设备十六进制代码。

(2) 指定到 netdevice 的虚拟功能（VF）的驱动器类型。

(3) 启用 RDMA 模式。

注意

如需了解 inSriovNetworkNodePolicy 的每个选项的详情，请参阅 Configuring SR-IOV network devices 部分。

当应用由 SriovNetworkNodePolicy 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。它可能需要几分钟时间来应用配置更改。确保集群中有足够的可用节点，用以预先处理被驱除的工作负载。

应用配置更新后，openshift-sriov-network-operator 命名空间中的所有 pod 将变为 Running 状态。

2. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3. 创建以下 SriovNetwork 对象，然后在 mlx-rdma-network.yaml 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- (1)
# ...
  vlan: <vlan>
  resourceName: mlxnics
```

(1) 为 ipam CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。

注意

如需 SriovNetwork 中的每个选项的详细说明，请参阅"Configuring SR-IOV additional network" 部分。

一个可选的库 app-netutil 提供了多种 API 方法来收集有关容器父 pod 的网络信息。

4. 运行以下命令来创建 SriovNetworkNodePolicy 对象：

```
$ oc create -f mlx-rdma-network.yaml
```

5. 创建以下 Pod spec，然后在 mlx-rdma-pod.yaml 文件中保存 YAML。

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "4" 5
        hugepages-1Gi: "4Gi" 6
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. 指定 target_namespace，它与 SriovNetwork 对象 mlx-rdma-network 创建于的命令空间相同。如果要在其他命名空间中创建 pod，在 Pod spec 和SriovNetwork 对象中更改 target_namespace。

2. 指定包含应用程序和应用程序使用的 RDMA 库的 RDMA 镜像。

3. 指定容器内的应用程序进行大页分配、系统资源分配和网络接口访问所需的额外功能。

4. 在 /mnt/huge 下将巨页卷挂载到 RDMA pod。巨页卷由 emptyDir 卷类型支持，medium 为Hugepages。

5. 指定 CPU 数量。RDMA pod 通常需要从 kubelet 分配专用 CPU。这可以通过将 CPU Manager 策略设置为 static，并创建带有有保障的 QoS 的 pod 来实现。

6. 指定巨页大小 hugepages-1Gi 或 hugepages-2Mi 以及分配给 RDMA pod 的巨页数量。单独配置 2Mi 和 1Gi 巨页。配置 1Gi 巨页需要在节点中添加内核参数。

6. 运行以下命令来创建 RDMA pod:

```
$ oc create -f mlx-rdma-pod.yaml
```

25.11.7. 在 OpenStack 上使用 OVS-DPDK 的集群测试 pod 模板

以下 testpmd pod 演示了使用巨页、保留 CPU 和 SR-IOV 端口创建容器。

testpmd pod 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-dpdk
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/dpdk1: 1 1
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
        openshift.io/dpdk1: 1
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  runtimeClassName: performance-cnf-performanceprofile 2
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 本例中名为 dpdk1 是一个用户创建的 SriovNetworkNodePolicy 资源。您可以为您创建的资源替换此名称。

2 如果您的性能配置集没有命名为 cnf-performance profile，请将该字符串替换为正确的性能配置集名称。

25.11.8. 在 OpenStack 上使用 OVS 硬件卸载的集群测试 pod 模板

以下 **testpmd** pod 在 Red Hat OpenStack Platform (RHOSP) 上演示了 Open vSwitch (OVS) 硬件卸载。

testpmd pod 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    k8s.v1.cni.cncf.io/networks: hwoffload1
spec:
  runtimeClassName: performance-cnf-performanceprofile (1)
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1) 如果您的性能配置集没有命名为 **cnf-performance profile**，请将该字符串替换为正确的性能配置集名称。

25.11.9. 其他资源

- 创建性能配置集
- 使用性能配置集调整 NIC 队列

- 置备实时和低延迟工作负载
- 安装 SR-IOV Network Operator
- 配置 SR-IOV 网络设备
- 使用 Whereabouts 进行动态 IP 地址分配配置
- 禁用单个 pod 的中断处理
- 配置 SR-IOV 以太网网络附加
- app-netutil 库提供了几个 API 方法，用于收集容器父 pod 的网络信息。

25.12. 使用 POD 级别绑定

在 pod 级别的绑定对于启用需要高可用性和更多吞吐量的 pod 内的工作负载至关重要。使用 pod 级别绑定，您可以在内核模式接口上从多个根 I/O 虚拟化(SR-IOV)虚拟功能接口创建绑定接口。SR-IOV 虚拟功能传递到 pod，并附加到内核驱动程序中。

需要 pod 级别绑定的一个场景是从不同物理功能的多个 SR-IOV 虚拟功能创建绑定接口。可以利用主机上的两个不同物理功能创建绑定接口，以便在 pod 级别上实现高可用性。

有关创建 SR-IOV 网络、网络策略、网络附加定义和 pod 等任务的指导，请参阅配置 SR-IOV 网络设备。

25.12.1. 从两个 SR-IOV 接口配置绑定接口

绑定可让多个网络接口聚合到一个逻辑 "bonded" 接口。绑定 Container Network Interface (Bond-CNI) 将绑定功能引入容器中。

Bond-CNI 可使用单根 I/O 虚拟化 (SR-IOV) 虚拟功能创建，并将它们放在容器网络命名空间中。

OpenShift Container Platform 仅支持使用 SR-IOV 虚拟功能的 Bond-CNI。SR-IOV Network Operator 提供了管理虚拟功能所需的 SR-IOV CNI 插件。不支持其他 CNI 或接口类型。

先决条件

- 必须安装 SR-IOV Network Operator，并配置为获取容器中的虚拟功能。
- 要配置 SR-IOV 接口，必须为每个接口创建一个 SR-IOV 网络和策略。
- SR-IOV Network Operator 根据定义的 SR-IOV 网络和策略，为每个 SR-IOV 接口创建一个网络附加定义。
- linkState 设置为 SR-IOV 虚拟功能的默认值auto。

25.12.1.1. 创建绑定网络附加定义

现在，SR-IOV 虚拟功能可用，您可以创建一个绑定网络附加定义。

```
apiVersion: "k8s.cni.cncf.io/v1"
  kind: NetworkAttachmentDefinition
  metadata:
    name: bond-net1
    namespace: demo
```

```
spec:
  config: '{
  "type": "bond", (1)
  "cniVersion": "0.3.1",
  "name": "bond-net1",
  "mode": "active-backup", (2)
  "failOverMac": 1, (3)
  "linksInContainer": true, (4)
  "miimon": "100",
  "mtu": 1500,
  "links": [ (5)
       {"name": "net1"},
       {"name": "net2"}
    ],
  "ipam": {
       "type": "host-local",
       "subnet": "10.56.217.0/24",
       "routes": [{
       "dst": "0.0.0.0/0"
       }],
       "gateway": "10.56.217.1"
    }
  }'
```

1. cni-type 始终设置为 **bond**。
2. **mode** 属性指定绑定模式。

注意

支持的绑定模式有：

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2

对于 **balance-rr** 或 **balance-xor** 模式，您必须为 SR-IOV 虚拟功能将 **trust** 模式设置为 **on**。

3. active-backup 模式的 **failover** 属性是必需的，必须设为 1。
4. **linksInContainer=true** 标志告知 Bond CNI 在容器内找到所需的接口。默认情况下，Bond CNI 会查找主机上的这些接口，该接口无法与 SRIOV 和 Multus 集成。
5. **links** 部分定义将用于创建绑定的接口。默认情况下，Multus 将附加的接口命名为 "net"，再加上一个连续的数字。

25.12.1.2. 使用绑定接口创建 pod

1. 使用名为 example **podbonding.yaml** 的 YAML 文件创建 pod 来测试设置，其内容类似以下示例：

```
apiVersion: v1
  kind: Pod
  metadata:
    name: bondpod1
    namespace: demo
    annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1 (1)
  spec:
    containers:
    - name: podexample
      image: quay.io/openshift/origin-network-interface-bond-cni:4.11.0
      command: ["/bin/bash", "-c", "sleep INF"]
```

(1) 注意网络注解：它包含两个 SR-IOV 网络附加，以及一个绑定网络附加。绑定附加使用两个 SR-IOV 接口作为绑定的端口接口。

2. 运行以下命令来应用 yaml:

```
$ oc apply -f podbonding.yaml
```

3. 使用以下命令检查 pod 接口：

```
$ oc rsh -n demo bondpod1
sh-4.4#
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen
1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
3: eth0@if150: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc
noqueue state UP
link/ether 62:b1:b5:c8:fb:7a brd ff:ff:ff:ff:ff:ff
inet 10.244.1.122/24 brd 10.244.1.255 scope global eth0
valid_lft forever preferred_lft forever
4: net3: <BROADCAST,MULTICAST,UP,LOWER_UP400> mtu 1500 qdisc noqueue
state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (1)
inet 10.56.217.66/24 scope global bond0
valid_lft forever preferred_lft forever
43: net1: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (2)
44: net2: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff (3)
```

(1) 绑定接口自动命名为 net3。要设置特定的接口名称，请将 @name 后缀添加到 pod 的 k8s.v1.cni.cncf.io/networks 注解。

(2) net1 接口基于 SR-IOV 虚拟功能。

(3) net2 接口基于 SR-IOV 虚拟功能。

注意

如果在 pod 注解中没有配置接口名称，接口名称会自动分配为 net<n>，其中 <n> 以 1 开始。

4. **可选： 如果要为 example bond0 设置一个特定的接口名称，请编辑 k8s.v1.cni.cncf.io/networks 注解，并将 bond0 设为接口名称，如下所示：**

```
annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-
net1@bond0
```

25.13. 配置硬件卸载 (OFFLOADING)

作为集群管理员，您可以在兼容节点上配置硬件卸载，以提高数据处理性能并减少主机 CPU 的负载。

25.13.1. 关于硬件卸载

Open vSwitch 硬件卸载是一种处理网络任务的方法，方法是将它们从 CPU 中分离出来，并将它们卸载到网络接口控制器上的专用处理器。因此，集群可从更快的数据传输速度、CPU 工作负载减少并降低计算成本中受益。

此功能的关键元素是网络接口控制器的现代类，称为 SmartNIC。SmartNIC 是一个网络接口控制器，它可以处理计算密集型网络处理任务。与专用图形卡可提高图形性能的方式相同，T SmartNIC 可改进网络性能。在各个情形中，专用处理器提高了特定类型的处理任务的性能。

在 OpenShift Container Platform 中，您可以为具有兼容 SmartNIC 的裸机节点配置硬件卸载。SR-IOV Network Operator 配置并启用硬件卸载。

硬件卸载并不适用于所有工作负载或应用程序类型。只支持以下两种通信类型：

- **pod 到 pod**
- **Pod 到服务，其中服务是一个由常规 pod 支持的 ClusterIP 服务**

在所有情况下，只有在将 pod 和服务分配给具有兼容 SmartNIC 的节点时，硬件卸载才会发生。假设节点上带有硬件卸载的 pod 会尝试与常规节点上的服务进行通信。常规节点上，所有处理都会在内核中进行，因此 pod 到服务通信的整体性能仅限于该常规节点的最大性能。硬件卸载与 DPDK 应用程序不兼容。

在节点上启用硬件卸载，但没有配置 pod 使用，可能会导致 pod 流量的吞吐量性能降低。您无法为 OpenShift Container Platform 管理的 pod 配置硬件卸载。

25.13.2. 支持的设备

在以下网络接口控制器上支持硬件卸载：

表 25.16. 支持的网络接口控制器

制造商	model	供应商 ID	设备 ID
Mellanox	MT27800 系列 [ConnectX-5]	15b3	1017
Mellanox	MT28880 系列 [ConnectX-5 Ex]	15b3	1019

制造商	model	供应商 ID	设备 ID
Mellanox	MT2892 系列 [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 系列 [ConnectX-6 Lx]	15b3	101f
Mellanox	ConnectX-6 NIC 模式中的 MT42822 BlueField-2	15b3	a2d6

25.13.3. 先决条件

- 集群至少有一个裸机带有网络接口控制器，支持进行硬件卸载。
- 已安装 SR-IOV Network Operator。
- 集群使用 OVN-Kubernetes 网络插件。
- 在 OVN-Kubernetes 网络插件配置中，gatewayConfig.routingViaHost 字段被设置为 false。

25.13.4. 为硬件卸载配置机器配置池

要启用硬件卸载，您必须首先创建一个专用的机器配置池，并将其配置为使用 SR-IOV Network Operator。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 为您要使用硬件卸载的机器创建机器配置池。
   a. 创建一个文件，如 mcp-offloading.yaml，其内容类似以下示例：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-offloading 1
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values:
[worker,mcp-offloading]} 2
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-offloading: "" 3
```

1 2 用于硬件卸载的机器配置池的名称。

3 此节点角色标签用于添加节点到机器配置池。

b. 应用机器配置池的配置：

```
$ oc create -f mcp-offloading.yaml
```

2. 将节点添加到机器配置池。使用池的节点角色标签标记每个节点：

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3. 可选：要验证是否创建了新池，请运行以下命令：

```
$ oc get nodes
```

输出示例

```
NAME      STATUS   ROLES                  AGE   VERSION
master-0  Ready    master                 2d    v1.29.4
master-1  Ready    master                 2d    v1.29.4
master-2  Ready    master                 2d    v1.29.4
worker-0  Ready    worker                 2d    v1.29.4
worker-1  Ready    worker                 2d    v1.29.4
worker-2  Ready    mcp-offloading,worker  47h   v1.29.4
worker-3  Ready    mcp-offloading,worker  47h   v1.29.4
```

4. 将此机器配置池添加到 SriovNetworkPoolConfig 自定义资源中：

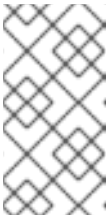
a. 创建一个文件，如 sriov-pool-config.yaml，其内容类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

1 用于硬件卸载的机器配置池的名称。

b. 应用配置：

```
$ oc create -f <SriovNetworkPoolConfig_name>.yaml
```



注意

当您应用由 SriovNetworkPoolConfig 对象中指定的配置时，SR-IOV Operator 会排空并重启机器配置池中的节点。

它可能需要几分钟时间来应用配置更改。

25.13.5. 配置 SR-IOV 网络节点策略

您可以通过创建 SR-IOV 网络节点策略来为节点创建 SR-IOV 网络设备配置。要启用硬件卸载，您必须使用值 "switchdev" 定义 .spec.eSwitchMode 字段。

以下流程为带有硬件卸载的网络接口控制器创建 SR-IOV 接口。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建一个文件，如 sriov-node-policy.yaml，其内容类似以下示例：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy (1)
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice (2)
  eSwitchMode: "switchdev" (3)
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 6
  priority: 5
  resourceName: mlxnics
```

(1) 自定义资源对象的名称。

(2) 必需。vfio-pci 不支持硬件卸载。

(3) 必需。

2. 应用策略的配置：

```
$ oc create -f sriov-node-policy.yaml
```

注意

当您应用由 SriovNetworkPoolConfig 对象中指定的配置时，SR-IOV Operator 会排空并重启机器配置池中的节点。

它可能需要几分钟时间来应用配置更改。

25.13.5.1. OpenStack 的 SR-IOV 网络节点策略示例

以下示例描述了在 Red Hat OpenStack Platform (RHOSP) 上使用硬件卸载的网络接口控制器 (NIC) 的 SR-IOV 接口。

用于 RHOSP 上带有硬件卸载的 NIC 的 SR-IOV 接口

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ${name}
  namespace: openshift-sriov-network-operator
spec:
  deviceType: switchdev
  isRdma: true
  nicSelector:
    netFilter: openstack/NetworkID:${net_id}
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: ${name}
```

25.13.6. 使用虚拟功能提高网络流量性能

按照以下步骤，为 OVN-Kubernetes 管理端口分配虚拟功能，并提高其网络流量性能。

此流程会导致创建两个池：第一个池具有 OVN-Kubernetes 使用的虚拟功能，第二个由剩余的虚拟功能组成。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 运行以下命令，将 network.operator.openshift.io/smart-nic 标签添加到带有 SmartNIC 的每个 worker 节点：

   ```
   $ oc label node <node-name> network.operator.openshift.io/smart-nic=
   ```

   使用 oc get nodes 命令获取可用节点的列表。

2. 为管理端口创建一个名为 sriov-node-mgmt-vf-policy.yaml 的策略，其内容如下：

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetworkNodePolicy
   metadata:
     name: sriov-node-mgmt-vf-policy
     namespace: openshift-sriov-network-operator
   spec:
     deviceType: netdevice
     eSwitchMode: "switchdev"
   ```

```
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0#0-0 1
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 2
  priority: 5
  resourceName: mgmtvf
```

1. 根据您的用例，将此设备替换为适当的网络设备。pfNames 值的 #0-0 部分保留了 OVN-Kubernetes 使用的一个虚拟功能。

2. 此处提供的值是一个示例。使用满足您的要求替换这个值。如需更多信息，请参阅*附加资源*部分中的 *SR-IOV 网络配置对象*。

3. 创建名为 sriov-node-policy.yaml 的策略，其内容如下：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0#1-5 1
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 2
  priority: 5
  resourceName: mlxnics
```

1. 根据您的用例，将此设备替换为适当的网络设备。

2. 此处提供的值是一个示例。使用 sriov-node-mgmt-vf-policy.yaml 文件中指定的值替换这个值。如需更多信息，请参阅*附加资源*部分中的 *SR-IOV 网络配置对象*。

注意

sriov-node-mgmt-vf-policy.yaml 文件具有与 sriov-node-policy.yaml 文件不同的 pfNames 和 resourceName 键的值。

4. 为这两个策略应用配置：

```
$ oc create -f sriov-node-policy.yaml
```

```
$ oc create -f sriov-node-mgmt-vf-policy.yaml
```

5. 在集群中创建 Cluster Network Operator (CNO) ConfigMap 以进行管理配置：

   a. 创建名为 **hardware-offload-config.yaml** 的 ConfigMap，其内容如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hardware-offload-config
  namespace: openshift-network-operator
data:
  mgmt-port-resource-name: openshift.io/mgmtvf
```

   b. 应用 ConfigMap 的配置：

```
$ oc create -f hardware-offload-config.yaml
```

其他资源

- SR-IOV 网络节点配置对象

25.13.7. 创建网络附加定义

在定义机器配置池和 SR-IOV 网络节点策略后，您可以为您指定的网络接口卡创建网络附加定义。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建一个文件，如 **net-attach-def.yaml**，其内容类似以下示例：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-attach-def 1
  namespace: net-attach-def 2
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnics 3
spec:
  config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-
overlay","ipam":{},"dns":{}}'
```

1. 网络附加定义的名称。
2. 网络附加定义的命名空间。

3 这是您在 SriovNetworkNodePolicy 对象中指定的 spec.resourceName 字段的值。

2. 应用网络附加定义的配置：

```
$ oc create -f net-attach-def.yaml
```

验证

- 运行以下命令，以查看是否存在新定义：

```
$ oc get net-attach-def -A
```

输出示例

```
NAMESPACE       NAME           AGE
net-attach-def  net-attach-def  43h
```

25.13.8. 在 pod 中添加网络附加定义

创建机器配置池后，SriovNetworkPoolConfig 和 SriovNetworkNodePolicy 自定义资源以及网络附加定义后，您可以通过在 pod 规格中添加网络附加定义来将这些配置应用到 pod。

流程

- 在 pod 规格中，添加 .metadata.annotations.k8s.v1.cni.cncf.io/networks 字段，并为硬件卸载指定您创建的网络附加定义：

```
....
metadata:
  annotations:
    v1.multus-cni.io/default-network: net-attach-def/net-attach-def 1
```

1 该值必须是您为硬件卸载而创建的网络附加定义的名称和命名空间。

25.14. 将 BLUEFIELD-2 从 DPU 切换到 NIC

您可以将 Bluefield-2 网络设备从数据处理单元 (DPU) 模式切换到网络接口控制器 (NIC) 模式。

25.14.1. 将 Bluefield-2 从 DPU 模式切换到 NIC 模式

使用以下步骤将 Bluefield-2 从数据处理单元 (DPU) 模式切换到网络接口控制器 (NIC) 模式。

重要

目前，只支持将 Bluefield-2 从 DPU 切换到 NIC 模式。不支持从 NIC 模式切换到 DPU 模式。

先决条件

- 已安装 SR-IOV Network Operator。如需更多信息，请参阅"安装 SR-IOV Network Operator"。
- 您已将 Bluefield-2 更新至最新的固件。如需更多信息，请参阅 NVIDIA BlueField-2 的固件。

流程

1. 输入以下命令为每个 worker 节点添加以下标签：

```
$ oc label node <example_node_name_one> node-role.kubernetes.io/sriov=
```

```
$ oc label node <example_node_name_two> node-role.kubernetes.io/sriov=
```

2. 为 SR-IOV Network Operator 创建机器配置池，例如：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: sriov
spec:
  machineConfigSelector:
    matchExpressions:
    - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,sriov]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/sriov: ""
```

3. 将以下 machineconfig.yaml 文件应用到 worker 节点：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: sriov
  name: 99-bf2-dpu
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ZmluZF9jb250YWluZXIoKSB7CiAgY3JpY3RsIHBzIC1vIGpzb24gfCBqcSAtciA
nLmNvbnRhaW5lcnNbXSB8IHNlbGVjdCgubWV0YWRhdGEubmFtZT09InNyaW92LW5ld
HdvcmstY29uZmlnLWRhZW1vbiIpIHwgLmlkJwp9CnVudGlsIG91dHB1dD0kKGZpbmRf
Y29udGFpbmVyKTsgW1sgLW4gIiRvdXRwdXQiF1dOyBkbwogIGVjaG8gIndhaXRpbmc
gZm9yIGNvbnRhaW5lciB0byBjb21lIHVwIgogIHNsZWVwIDE7CmRvbmUKISBzdWRvIG
NyaWN0bCBleGVjICRvdXRwdXQgL2JpbmRhdGEvc2NyaXB0cy9iZjItc3dpdGNoLW1vZ
GUuc2ggliRAIgo=
        mode: 0755
        overwrite: true
        path: /etc/default/switch_in_sriov_config_daemon.sh
    systemd:
      units:
```

```
    - name: dpu-switch.service
      enabled: true
      contents: |
        [Unit]
        Description=Switch BlueField2 card to NIC/DPU mode
        RequiresMountsFor=%t/containers
        Wants=network.target
        After=network-online.target kubelet.service
        [Service]
        SuccessExitStatus=0 120
        RemainAfterExit=True
        ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic ||
shutdown -r now' (1)
        Type=oneshot
        [Install]
        WantedBy=multi-user.target
```

(1) 可选：可以选择性地指定特定卡的 PCI 地址，如 ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'。默认情况下会选择第一个设备。如果有多个设备，您必须指定要使用的 PCI 地址。在将 Bluefield-2 从 DPU 模式切换到 NIC 模式的所有节点上，PCI 地址必须相同。

4. 等待 worker 节点重启。重启后，worker 节点上的 Bluefield-2 网络设备切换到 NIC 模式。

其他资源

安装 SR-IOV Network Operator

25.15. 卸载 SR-IOV NETWORK OPERATOR

要卸载 SR-IOV Network Operator，您必须删除所有正在运行的 SR-IOV 工作负载，卸载 Operator，并删除 Operator 使用的 webhook。

25.15.1. 卸载 SR-IOV Network Operator

作为集群管理员，您可以卸载 SR-IOV Network Operator。

先决条件

- 可以使用具有 cluster-admin 权限的账户访问 OpenShift Container Platform 集群。
- 已安装 SR-IOV Network Operator。

流程

1. 删除所有 SR-IOV 自定义资源(CR)：

```
$ oc delete sriovnetwork -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovnetworknodepolicy -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovibnetwork -n openshift-sriov-network-operator --all
```

2. 按照 "Deleting Operators from a cluster" 部分的说明从集群中移除 SR-IOV Network Operator。
3. 卸载 SR-IOV Network Operator 后，删除在集群中保留的 SR-IOV 自定义资源定义：

```
$ oc delete crd sriovibnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
```

4. 删除 SR-IOV Webhook：

```
$ oc delete mutatingwebhookconfigurations network-resources-injector-config
```

```
$ oc delete MutatingWebhookConfiguration sriov-operator-webhook-config
```

```
$ oc delete ValidatingWebhookConfiguration sriov-operator-webhook-config
```

5. 删除 SR-IOV Network Operator 命名空间：

```
$ oc delete namespace openshift-sriov-network-operator
```

其他资源

- 从集群中删除 Operator

第 26 章 OVN-KUBERNETES 网络插件

26.1. 关于 OVN-KUBERNETES 网络插件

OpenShift Container Platform 集群在 pod 和服务网络中使用虚拟网络。

Red Hat OpenShift Networking 的一部分，OVN-Kubernetes 网络插件是 OpenShift Container Platform 的默认网络供应商。OVN-Kubernetes 基于 Open Virtual Network（OVN），它提供了一个基于 overlay 的网络实现。使用 OVN-Kubernetes 插件的集群还在每个节点上运行 Open vSwitch (OVS)。OVN 在每个节点上配置 OVS 来实现声明的网络配置。

注意

OVN-Kubernetes 是 OpenShift Container Platform 和单节点 OpenShift 部署的默认网络解决方案。

OVN-Kubernetes （来自 OVS 项目）使用许多相同的结构，如开放流规则，以确定数据包通过网络传输的方式。如需更多信息，请参阅 Open Virtual Network 网站。

OVN-Kubernetes 是 OVS 的一系列守护进程，用于将虚拟网络配置转换为 OpenFlow 规则。OpenFlow 是一种用于与网络交换机和路由器通信的协议，为远程控制网络设备上的网络流量流提供了方法，让网络管理员能够配置、管理和监控网络流量的流。

OVN-Kubernetes 提供了 OpenFlow 提供的更多高级功能。OVN 支持分布式虚拟路由、分布式逻辑交换机、访问控制、DHCP 和 DNS。OVN 在逻辑流中实施分布式虚拟路由，这些路由等同于开放流。例如，如果您有一个 pod 在网络上发送 DHCP 请求，它会发送出该广播查找 DHCP 地址的逻辑流规则，该逻辑流规则与该数据包匹配，并且响应了网关，DNS 服务器是 IP 地址等。

OVN-Kubernetes 在每个节点上运行一个守护进程。数据库和 OVN 控制器都有守护进程集，每个节点上运行的 OVN 控制器。OVN 控制器在节点上对 Open vSwitch 守护进程进行编程，以支持网络提供程序功能；出口 IP、防火墙、路由器、混合网络、IPSEC 加密、IPv6 加密、网络策略日志、网络策略日志、硬件卸载和多播。

26.1.1. OVN-Kubernetes 目的

OVN-Kubernetes 网络插件是一个开源、功能齐全的 Kubernetes CNI 插件，它使用 Open Virtual Network (OVN)来管理网络流量。OVN 是一个社区开发、与供应商无关的网络虚拟化解决方案。OVN-Kubernetes 网络插件：

- 使用 OVN（开源虚拟网络）管理网络流量。OVN 是一个社区开发、与供应商无关的网络虚拟化解决方案。
- 实现 Kubernetes 网络策略支持，包括入口和出口规则。
- 使用 Geneve（通用网络虚拟化封装）协议而不是 VXLAN 在节点间创建覆盖网络。

OVN-Kubernetes 网络插件比 OpenShift SDN 提供以下优点。

- 在支持的平台上完全支持 IPv6 单堆栈和 IPv4/IPv6 双栈网络
- 支持 Linux 和 Microsoft Windows 工作负载中的混合集群
- 可选的、集群内通信的 IPsec 加密
- 将网络数据处理从主机 CPU 卸载到兼容的网卡和数据处理单元 (DPU)

26.1.2. 支持的网络插件功能列表

Red Hat OpenShift Networking 为网络插件(OpenShift SDN 和 OVN-Kubernetes)提供了两个选项，用于网络插件。下表总结了这两个网络插件的当前功能支持：

表 26.1. 默认 CNI 网络插件功能比较

功能	OVN-Kubernetes	OpenShift SDN
出口 IP	支持	支持
Egress 防火墙 [1]	支持	支持
出口路由器	支持 [2]	支持
混合网络	支持	不支持
集群内通信的 IPsec 加密	支持	不支持
IPv6	支持 [3] [4]	不支持
Kubernetes 网络策略	支持	支持
Kubernetes 网络策略日志	支持	不支持
硬件卸载	支持	不支持
多播	支持	支持

1. 在 OpenShift SDN 中，出口防火墙也称为出口网络策略。这和网络策略出口不同。
2. OVN-Kubernetes 的出口路由器仅支持重定向模式。
3. 只有裸机、vSphere、IBM Power®、IBM Z® 和 Red Hat OpenStack 集群才支持 IPv6。
4. IBM Power®、IBM Z® 和 Red Hat OpenStack 集群不支持 IPv6 单堆栈。

26.1.3. OVN-Kubernetes IPv6 和双栈限制

OVN-Kubernetes 网络插件有以下限制：

- 对于为双栈网络配置的集群，IPv4 和 IPv6 流量都必须使用与默认网关相同的网络接口。如果不满足此要求，则 ovnkube-node 守护进程集中的主机上的容器集进入 CrashLoopBackOff 状态。如果您使用 oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml 等命令显示 pod，则 status 字段包含多个有关默认网关的消息，如以下输出所示：

```
I1006 16:09:50.985852   60651 helper_linux.go:73] Found default gateway interface br-ex 192.168.127.1
I1006 16:09:50.985923   60651 helper_linux.go:73] Found default gateway interface
```

```
ens4 fe80::5054:ff:febe:bcd4
F1006 16:09:50.985939   60651 ovnkube.go:130] multiple gateway interfaces detected:
br-ex ens4
```

唯一的解析是重新配置主机网络，以便两个 IP 系列都针对默认网关使用相同的网络接口。

- 对于为双栈网络配置的集群，IPv4 和 IPv6 路由表必须包含默认网关。如果不满足此要求，则 ovnkube-node 守护进程集中的主机上的容器集进入 CrashLoopBackOff 状态。如果您使用 oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml 等命令显示 pod，则 status 字段包含多个有关默认网关的消息，如以下输出所示：

```
I0512 19:07:17.589083  108432 helper_linux.go:74] Found default gateway interface br-
ex 192.168.123.1
F0512 19:07:17.589141  108432 ovnkube.go:133] failed to get default gateway interface
```

唯一的解析是重新配置主机网络，以便两个 IP 系列都包含默认网关。

26.1.4. 会话关联性

会话关联性是适用于 Kubernetes Service 对象的功能。如果要确保每次连接到 <service_VIP>:<Port> 时，您可以使用 *会话关联性*，流量始终被加载到同一后端。如需更多信息，包括如何根据客户端的 IP 地址设置会话关联性，请参阅会话关联性。

会话关联性的粘性超时
OpenShift Container Platform 的 OVN-Kubernetes 网络插件根据最后一个数据包计算来自客户端的会话的粘性超时。例如，如果您运行 curl 命令 10 次，则粘性会话计时器从第十个数据包开始，而不是第一个数据包。因此，如果客户端不断联系该服务，则会话永远不会超时。当服务没有收到 timeoutSeconds 参数所设定的时间的数据包时，超时开始。

其他资源

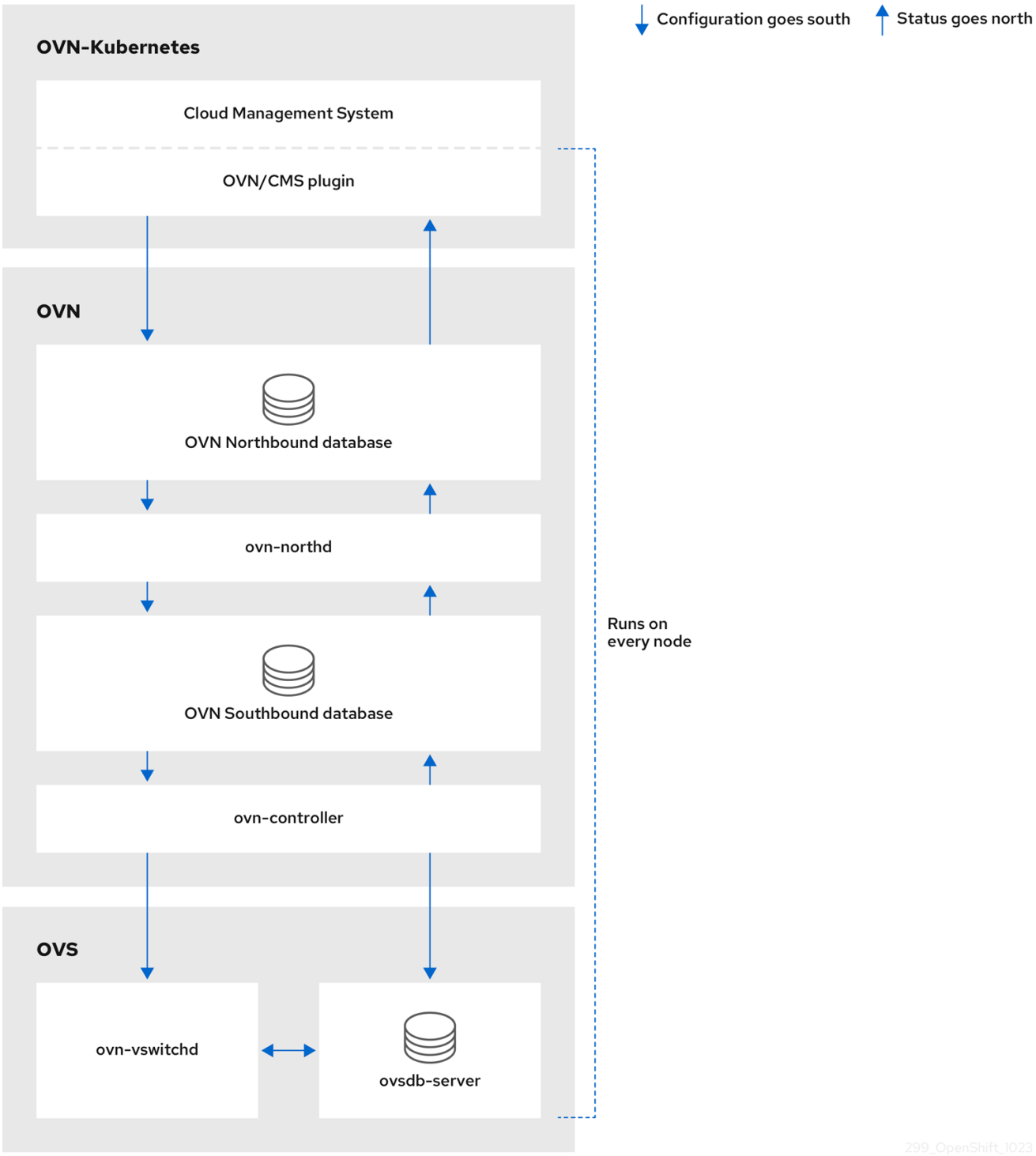
- 为项目配置出口防火墙
- 关于网络策略
- 记录网络策略事件
- 为项目启用多播
- 配置 IPsec 加密
- Network [operator.openshift.io/v1]

26.2. OVN-KUBERNETES 架构

26.2.1. OVN-Kubernetes 架构简介

下图显示了 OVN-Kubernetes 架构。

图 26.1. OVK-Kubernetes 架构



主要组件是：

- Cloud Management System (CMS) - OVN 的特定平台客户端，为 OVN 集成提供 CMS 特定的插件。该插件将云管理系统的逻辑网络配置概念转换为 OVN 理解的 CMS 配置数据库中。
- OVN 北向数据库(nbdb)容器 - 存储由 CMS 插件传递的逻辑网络配置。
- OVN 南向数据库(sbdb)容器 - 存储每个节点上的 Open vSwitch (OVS)系统的物理和逻辑网络配置状态，包括绑定它们的表。

- OVN 北向守护进程 (ovn-northd) - 这是 nbdb 容器和 sbdb 容器之间的中介客户端。它以传统网络概念的形式将逻辑网络配置（从 nbdb 容器）转换为 sbdb 容器中的逻辑数据路径流。ovn-northd 守护进程的容器名称为 northd，它在 ovnkube-node pod 中运行。

- ovn-controller - 这是与 OVS 和 hypervisor 交互的 OVN 代理，适用于sbdb 容器所需的任何信息或更新。ovn-controller 从 sbdb 容器读取逻辑流，将它们转换为 OpenFlow 流，并将它们发送到节点的 OVS 守护进程。容器名称为 ovn-controller，它在 ovnkube-node pod 中运行。

OVN 北向数据库、北向数据库和南向数据库在集群中的每个节点上运行，大部分包含对该节点本地的和进程信息。

OVN 北向数据库具有通过云管理系统(CMS)传递到它的逻辑网络配置。OVN 北向数据库包含网络的当前状态，以逻辑端口、逻辑交换机、逻辑路由器等形式显示。ovn-northd (northd 容器) 连接到 OVN 北向数据库和 OVN 南向数据库。它以传统网络概念的形式将逻辑网络配置转换为 OVN 北向数据库中的逻辑数据路径流。

OVN 南向数据库具有网络的物理和逻辑表示，并将它们连接在一起。它包含节点的机箱信息，以及其他结构，如连接到集群中其他节点所需的远程传输交换机端口。OVN 南向数据库还包含所有逻辑流。逻辑流与每个节点上运行的 ovn-controller 进程共享，ovn-controller 将它们转换为 OpenFlow 规则到程序 Open vSwitch(OVS)。

Kubernetes control plane 节点在单独的节点上包含两个 ovnkube-control-plane pod，它为集群中的每个节点执行中央 IP 地址管理 (IPAM) 分配。在任何给定时间，只有一个 ovnkube-control-plane pod 是领导。

26.2.2. 列出 OVN-Kubernetes 项目中的所有资源

查找在 OVN-Kubernetes 项目中运行的资源和容器对于帮助您了解 OVN-Kubernetes 网络实施非常重要。

先决条件

- 使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。

流程

1. 运行以下命令，以获取 OVN-Kubernetes 项目中的所有资源、端点和 ConfigMap ：

```
$ oc get all,ep,cm -n openshift-ovn-kubernetes
```

输出示例

```
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
NAME                                  READY   STATUS    RESTARTS      AGE
pod/ovnkube-control-plane-65c6f55656-6d55h   2/2     Running   0             114m
pod/ovnkube-control-plane-65c6f55656-fd7vw   2/2     Running   2 (104m ago)   114m
pod/ovnkube-node-bcvts                8/8     Running   0             113m
pod/ovnkube-node-drgvv                8/8     Running   0             113m
pod/ovnkube-node-f2pxt                8/8     Running   0             113m
pod/ovnkube-node-frqsb                8/8     Running   0             105m
pod/ovnkube-node-lbxkk                8/8     Running   0             105m
pod/ovnkube-node-tt7bx                8/8     Running   1 (102m ago)   105m
```

```
NAME                                   TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
service/ovn-kubernetes-control-plane   ClusterIP   None         <none>        9108/TCP
114m
service/ovn-kubernetes-node            ClusterIP   None         <none>
9103/TCP,9105/TCP   114m

NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR              AGE
daemonset.apps/ovnkube-node   6         6         6       6            6
beta.kubernetes.io/os=linux   114m

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ovnkube-control-plane   3/3     3            3           114m

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/ovnkube-control-plane-65c6f55656   3         3         3       114m

NAME                                   ENDPOINTS                                         AGE
endpoints/ovn-kubernetes-control-plane   10.0.0.3:9108,10.0.0.4:9108,10.0.0.5:9108
114m
endpoints/ovn-kubernetes-node            10.0.0.3:9105,10.0.0.4:9105,10.0.0.5:9105 + 9
more...   114m

NAME                            DATA   AGE
configmap/control-plane-status      1      113m
configmap/kube-root-ca.crt          1      114m
configmap/openshift-service-ca.crt   1      114m
configmap/ovn-ca                    1      114m
configmap/ovnkube-config            1      114m
configmap/signer-ca                 1      114m
```

集群中的每个节点都有一个 ovnkube-node pod。ovnkube-config 配置映射具有 OpenShift Container Platform OVN-Kubernetes 配置。

2. 运行以下命令，列出 ovnkube-node pod 中的所有容器：

```
$ oc get pods ovnkube-node-bcvts -o jsonpath='{.spec.containers[*].name}' -n
openshift-ovn-kubernetes
```

预期输出

```
ovn-controller ovn-acl-logging kube-rbac-proxy-node kube-rbac-proxy-ovn-metrics
northd nbdb sbdb ovnkube-controller
```

ovnkube-node pod 由几个容器组成。它负责托管北向数据库(nbdb 容器)、南向数据库(sbdb 容器)、北守护进程 (northd 容器)、ovn-controller 和 ovnkube-controller 容器。ovnkube-controller 容器会监视 API 对象，如 pod、egress IP、命名空间、服务、端点、出口防火墙和网络策略。它还负责为该节点从可用子网池中分配 pod IP。

3. 运行以下命令，列出 ovnkube-control-plane pod 中的所有容器：

```
$ oc get pods ovnkube-control-plane-65c6f55656-6d55h -o
jsonpath='{.spec.containers[*].name}' -n openshift-ovn-kubernetes
```

预期输出

```
kube-rbac-proxy ovnkube-cluster-manager
```

ovnkube-control-plane pod 有一个容器(ovnkube-cluster-manager)，它驻留在每个 OpenShift Container Platform 节点上。ovnkube-cluster-manager 容器分配 pod 子网，将子网 IP 传输交换机到集群中的每个节点。kube-rbac-proxy 容器监控 ovnkube-cluster-manager 容器的指标。

26.2.3. 列出 OVN-Kubernetes 北向数据库内容

每个节点都由该节点上 ovnkube-node pod 中运行的 ovnkube-controller 容器控制。若要了解 OVN 逻辑网络实体，您需要检查作为容器在该节点上的 ovnkube-node pod 中运行的北向数据库，以查看您要查看的节点中的对象。

先决条件

- 使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。

流程

要在集群中运行 ovn nbctl 或 sbctl 命令，您必须在相关节点上的 nbdb 或 sbdb 容器中打开远程 shell

1. 运行以下命令列出 pod：

```
$ oc get po -n openshift-ovn-kubernetes
```

输出示例

```
NAME                                  READY   STATUS    RESTARTS      AGE
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m
ovnkube-node-55xs2                       8/8     Running   0             26m
ovnkube-node-7r84r                       8/8     Running   0             16m
ovnkube-node-bqq8p                       8/8     Running   0             17m
ovnkube-node-mkj4f                       8/8     Running   0             26m
ovnkube-node-mlr8k                       8/8     Running   0             26m
ovnkube-node-wqn2m                       8/8     Running   0             16m
```

2. 可选：要使用节点信息列出 pod，请运行以下命令：

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

输出示例

```
NAME                                  READY   STATUS    RESTARTS      AGE   IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m   10.0.0.3     ci-
ln-t487nnb-72292-mdcnq-master-1   <none>           <none>
```

```
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0          27m   10.0.0.4     ci-
ln-t487nnb-72292-mdcnq-master-2   <none>          <none>
ovnkube-node-55xs2                 8/8     Running   0          26m   10.0.0.4     ci-ln-
t487nnb-72292-mdcnq-master-2   <none>          <none>
ovnkube-node-7r84r                 8/8     Running   0          17m   10.0.128.3   ci-ln-
t487nnb-72292-mdcnq-worker-b-wbz7z   <none>          <none>
ovnkube-node-bqq8p                 8/8     Running   0          17m   10.0.128.2   ci-ln-
t487nnb-72292-mdcnq-worker-a-lh7ms   <none>          <none>
ovnkube-node-mkj4f                 8/8     Running   0          27m   10.0.0.5     ci-ln-
t487nnb-72292-mdcnq-master-0   <none>          <none>
ovnkube-node-mlr8k                 8/8     Running   0          27m   10.0.0.3     ci-ln-
t487nnb-72292-mdcnq-master-1   <none>          <none>
ovnkube-node-wqn2m                 8/8     Running   0          17m   10.0.128.4   ci-ln-
t487nnb-72292-mdcnq-worker-c-przlm   <none>          <none>
```

3. 运行以下命令，进入 pod 以查看北向数据库：

```
$ oc rsh -c nbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. 运行以下命令以显示北向数据库中的所有对象：

```
$ ovn-nbctl show
```

此处列出输出太长。列表中包含 NAT 规则、逻辑交换机、负载均衡器等。

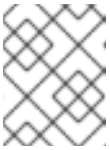
您可以使用以下可选命令缩小并专注于特定组件：

a. 运行以下命令以显示逻辑路由器列表：

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c northd -- ovn-nbctl lr-list
```

输出示例

```
45339f4f-7d0b-41d0-b5f9-9fca9ce40ce6 (GR_ci-ln-t487nnb-72292-mdcnq-master-2)
96a0a0f0-e7ed-4fec-8393-3195563de1b8 (ovn_cluster_router)
```



注意

在这个输出中，您可以看到每个节点中存在路由器，再加上 ovn_cluster_router。

b. 运行以下命令以显示逻辑交换机列表：

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl ls-list
```

输出示例

```
bdd7dc3d-d848-4a74-b293-cc15128ea614 (ci-ln-t487nnb-72292-mdcnq-master-2)
b349292d-ee03-4914-935f-1940b6cb91e5 (ext_ci-ln-t487nnb-72292-mdcnq-master-
2)
```

```
0aac0754-ea32-4e33-b086-35eeabf0a140 (join)
992509d7-2c3f-4432-88db-c179e43592e5 (transit_switch)
```

注意

在这个输出中，您可以看到每个节点的 ext 交换机以及节点名称本身和 join 开关。

c. 运行以下命令以显示负载均衡器列表：

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl lb-list
```

输出示例

```
UUID                                    LB                  PROTO      VIP                    IPs
7c84c673-ed2a-4436-9a1f-9bc5dd181eea    Service_default/    tcp
172.30.0.1:443          10.0.0.3:6443,169.254.169.2:6443,10.0.0.5:6443
4d663fd9-ddc8-4271-b333-4c0e279e20bb    Service_default/    tcp
172.30.0.1:443          10.0.0.3:6443,10.0.0.4:6443,10.0.0.5:6443
292eb07f-b82f-4962-868a-4f541d250bca    Service_openshif    tcp
172.30.105.247:443      10.129.0.12:8443
034b5a7f-bb6a-45e9-8e6d-573a82dc5ee3    Service_openshif    tcp
172.30.192.38:443       10.0.0.3:10259,10.0.0.4:10259,10.0.0.5:10259
a68bb53e-be84-48df-bd38-bdd82fcd4026    Service_openshif    tcp
172.30.161.125:8443     10.129.0.32:8443
6cc21b3d-2c54-4c94-8ff5-d8e017269c2e    Service_openshif    tcp
172.30.3.144:443        10.129.0.22:8443
37996ffd-7268-4862-a27f-61cd62e09c32    Service_openshif    tcp
172.30.181.107:443      10.129.0.18:8443
81d4da3c-f811-411f-ae0c-bc6713d0861d    Service_openshif    tcp
172.30.228.23:443       10.129.0.29:8443
ac5a4f3b-b6ba-4ceb-82d0-d84f2c41306e    Service_openshif    tcp
172.30.14.240:9443      10.129.0.36:9443
c88979fb-1ef5-414b-90ac-43b579351ac9    Service_openshif    tcp
172.30.231.192:9001
10.128.0.5:9001,10.128.2.5:9001,10.129.0.5:9001,10.129.2.4:9001,10.130.0.3:9001,10.1
31.0.3:9001
fcb0a3fb-4a77-4230-a84a-be45dce757e8    Service_openshif    tcp
172.30.189.92:443       10.130.0.17:8440
67ef3e7b-ceb9-4bf0-8d96-b43bde4c9151    Service_openshif    tcp
172.30.67.218:443       10.129.0.9:8443
d0032fba-7d5e-424a-af25-4ab9b5d46e81    Service_openshif    tcp
172.30.102.137:2379     10.0.0.3:2379,10.0.0.4:2379,10.0.0.5:2379
                                        tcp         172.30.102.137:9979
10.0.0.3:9979,10.0.0.4:9979,10.0.0.5:9979
7361c537-3eec-4e6c-bc0c-0522d182abd4    Service_openshif    tcp
172.30.198.215:9001
10.0.0.3:9001,10.0.0.4:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.128.4:
9001
0296c437-1259-410b-a6fd-81c310ad0af5    Service_openshif    tcp
172.30.198.215:9001
10.0.0.3:9001,169.254.169.2:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.
128.4:9001
```

```
5d5679f5-45b8-479d-9f7c-08b123c688b8    Service_openshif    tcp
172.30.38.253:17698     10.128.0.52:17698,10.129.0.84:17698,10.130.0.60:17698
2adcbab4-d1c9-447d-9573-b5dc9f2efbfa    Service_openshif    tcp
172.30.148.52:443       10.0.0.4:9202,10.0.0.5:9202
                                        tcp       172.30.148.52:444
10.0.0.4:9203,10.0.0.5:9203
                                        tcp       172.30.148.52:445
10.0.0.4:9204,10.0.0.5:9204
                                        tcp       172.30.148.52:446
10.0.0.4:9205,10.0.0.5:9205
2a33a6d7-af1b-4892-87cc-326a380b809b    Service_openshif    tcp
172.30.67.219:9091      10.129.2.16:9091,10.131.0.16:9091
                                        tcp       172.30.67.219:9092
10.129.2.16:9092,10.131.0.16:9092
                                        tcp       172.30.67.219:9093
10.129.2.16:9093,10.131.0.16:9093
                                        tcp       172.30.67.219:9094
10.129.2.16:9094,10.131.0.16:9094
f56f59d7-231a-4974-99b3-792e2741ec8d    Service_openshif    tcp
172.30.89.212:443       10.128.0.41:8443,10.129.0.68:8443,10.130.0.44:8443
08c2c6d7-d217-4b96-b5d8-c80c4e258116    Service_openshif    tcp
172.30.102.137:2379     10.0.0.3:2379,169.254.169.2:2379,10.0.0.5:2379
                                        tcp       172.30.102.137:9979
10.0.0.3:9979,169.254.169.2:9979,10.0.0.5:9979
60a69c56-fc6a-4de6-bd88-3f2af5ba5665    Service_openshif    tcp
172.30.10.193:443       10.129.0.25:8443
ab1ef694-0826-4671-a22c-565fc2d282ec    Service_openshif    tcp
172.30.196.123:443      10.128.0.33:8443,10.129.0.64:8443,10.130.0.37:8443
b1fb34d3-0944-4770-9ee3-2683e7a630e2    Service_openshif    tcp
172.30.158.93:8443      10.129.0.13:8443
95811c11-56e2-4877-be1e-c78ccb3a82a9    Service_openshif    tcp
172.30.46.85:9001       10.130.0.16:9001
4baba1d1-b873-4535-884c-3f6fc07a50fd    Service_openshif    tcp
172.30.28.87:443        10.129.0.26:8443
6c2e1c90-f0ca-484e-8a8e-40e71442110a    Service_openshif    udp
172.30.0.10:53
10.128.0.13:5353,10.128.2.6:5353,10.129.0.39:5353,10.129.2.6:5353,10.130.0.11:5353,
10.131.0.9:5353
```

注意

在这个截断的输出中，您可以看到有许多 OVN-Kubernetes 负载均衡器。OVN-Kubernetes 中的负载均衡器是服务的表示。

5. 运行以下命令，以显示可用于命令 **ovn-nbctl** 的选项：

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb ovn-nbctl --help
```

26.2.4. ovn-nbctl 的命令行参数检查北向数据库内容

下表描述了可与 **ovn-nbctl** 一起使用的命令行参数，以检查北向数据库的内容。

注意

在您要查看内容的 pod 中打开一个远程 shell，然后运行 **ovn-nbctl** 命令。

表 26.2. 检查北向数据库内容的命令行参数

参数	描述
ovn-nbctl show	从特定节点看到的北向数据库内容概述。
ovn-nbctl show <switch_or_router>	显示与指定交换机或路由器关联的详细信息。
ovn-nbctl lr-list	显示逻辑路由器。
ovn-nbctl lrp-list <router>	使用 **ovn-nbctl lr-list** 中的路由器信息来显示路由器端口。
ovn-nbctl lr-nat-list <router>	显示指定路由器的网络地址转换详情。
ovn-nbctl ls-list	显示逻辑交换机
ovn-nbctl lsp-list <switch>	使用 **ovn-nbctl ls-list** 的交换机信息来显示交换机端口。
ovn-nbctl lsp-get-type <port>	获取逻辑端口的类型。
ovn-nbctl lb-list	显示负载平衡器。

26.2.5. 列出 OVN-Kubernetes 南向数据库内容

每个节点都由该节点上 **ovnkube-node** pod 中运行的 **ovnkube-controller** 容器控制。若要了解 OVN 逻辑网络实体，您需要检查作为容器在该节点上的 **ovnkube-node** pod 中运行的北向数据库，以查看您要查看的节点中的对象。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI（**oc**）。

流程

要在集群中运行 ovn **nbctl** 或 **sbctl** 命令，您必须在相关节点上的 **nbdb** 或 **sbdb** 容器中打开远程 shell

1. 运行以下命令列出 pod：

```
$ oc get po -n openshift-ovn-kubernetes
```

输出示例

```
NAME                                  READY   STATUS    RESTARTS      AGE
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m
ovnkube-node-55xs2                    8/8     Running   0             26m
ovnkube-node-7r84r                    8/8     Running   0             16m
ovnkube-node-bqq8p                    8/8     Running   0             17m
ovnkube-node-mkj4f                    8/8     Running   0             26m
ovnkube-node-mlr8k                    8/8     Running   0             26m
ovnkube-node-wqn2m                    8/8     Running   0             16m
```

2. 可选：要使用节点信息列出 pod，请运行以下命令：

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

输出示例

```
NAME                                  READY   STATUS    RESTARTS      AGE   IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m   10.0.0.3     ci-
ln-t487nnb-72292-mdcnq-master-1   <none>           <none>
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m   10.0.0.4     ci-
ln-t487nnb-72292-mdcnq-master-2   <none>           <none>
ovnkube-node-55xs2                    8/8     Running   0             26m   10.0.0.4     ci-ln-
t487nnb-72292-mdcnq-master-2   <none>           <none>
ovnkube-node-7r84r                    8/8     Running   0             17m   10.0.128.3   ci-ln-
t487nnb-72292-mdcnq-worker-b-wbz7z   <none>           <none>
ovnkube-node-bqq8p                    8/8     Running   0             17m   10.0.128.2   ci-ln-
t487nnb-72292-mdcnq-worker-a-lh7ms   <none>           <none>
ovnkube-node-mkj4f                    8/8     Running   0             27m   10.0.0.5     ci-ln-
t487nnb-72292-mdcnq-master-0   <none>           <none>
ovnkube-node-mlr8k                    8/8     Running   0             27m   10.0.0.3     ci-ln-
t487nnb-72292-mdcnq-master-1   <none>           <none>
ovnkube-node-wqn2m                    8/8     Running   0             17m   10.0.128.4   ci-ln-
t487nnb-72292-mdcnq-worker-c-przlm   <none>           <none>
```

3. 进入 pod 以查看南向数据库：

```
$ oc rsh -c sbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. 运行以下命令以显示南向数据库中的所有对象：

```
$ ovn-sbctl show
```

输出示例

```
Chassis "5db31703-35e9-413b-8cdf-69e7eecb41f7"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-a-8bmwz
    Encap geneve
        ip: "10.0.128.4"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-worker-a-8bmwz
```

```
Chassis "070debed-99b7-4bce-b17d-17e720b7f8bc"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Encap geneve
        ip: "10.0.128.2"
        options: {csum="true"}
    Port_Binding k8s-ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding rtoe-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-monitoring_alertmanager-main-1
    Port_Binding rtoj-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding etor-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding cr-rtos-ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-e2e-loki_loki-promtail-qcrcz
    Port_Binding jtor-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-multus_network-metrics-daemon-mkd4t
    Port_Binding openshift-ingress-canary_ingress-canary-xtvj4
    Port_Binding openshift-ingress_router-default-6c76cbc498-pvlqk
    Port_Binding openshift-dns_dns-default-zz582
    Port_Binding openshift-monitoring_thanos-querier-57585899f5-lbf4f
    Port_Binding openshift-network-diagnostics_network-check-target-tn228
    Port_Binding openshift-monitoring_prometheus-k8s-0
    Port_Binding openshift-image-registry_image-registry-68899bd877-xqxjj
Chassis "179ba069-0af1-401c-b044-e5ba90f60fea"
    hostname: ci-ln-9gp362t-72292-v2p94-master-0
    Encap geneve
        ip: "10.0.0.5"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-0
Chassis "68c954f2-5a76-47be-9e84-1cb13bd9dab9"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-c-mjf9w
    Encap geneve
        ip: "10.0.128.3"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-worker-c-mjf9w
Chassis "2de65d9e-9abf-4b6e-a51d-a1e038b4d8af"
    hostname: ci-ln-9gp362t-72292-v2p94-master-2
    Encap geneve
        ip: "10.0.0.4"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-2
Chassis "1d371cb8-5e21-44fd-9025-c4b162cc4247"
    hostname: ci-ln-9gp362t-72292-v2p94-master-1
    Encap geneve
        ip: "10.0.0.3"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-1
```

此详细输出显示了附加到机箱的机箱和端口，本例中为所有路由器端口以及像主机网络一样运行的任何内容。任何 pod 使用源网络地址转换(SNAT)与更广泛的网络通信。其 IP 地址转换为运行 Pod 的节点的 IP 地址，然后发送到网络。

除了机箱信息外，南向数据库还具有所有逻辑流，这些逻辑流随后发送到每个节点上运行的 ovn-controller。ovn-controller 将逻辑流转换为开放流规则，最终程序 OpenvSwitch 以便您的 pod 可以遵循开放流规则，并使其从网络移出。

5. 运行以下命令以 ovn-sbctl 命令显示可用的选项：

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c sbdb ovn-sbctl --help
```

26.2.6. ovn-sbctl 的命令行参数，以检查南向数据库内容

下表描述了可用于 ovn-sbctl 的命令行参数，以检查南向数据库的内容。

注意

在您要查看内容的 pod 中打开一个远程 shell，然后运行 ovn-sbctl 命令。

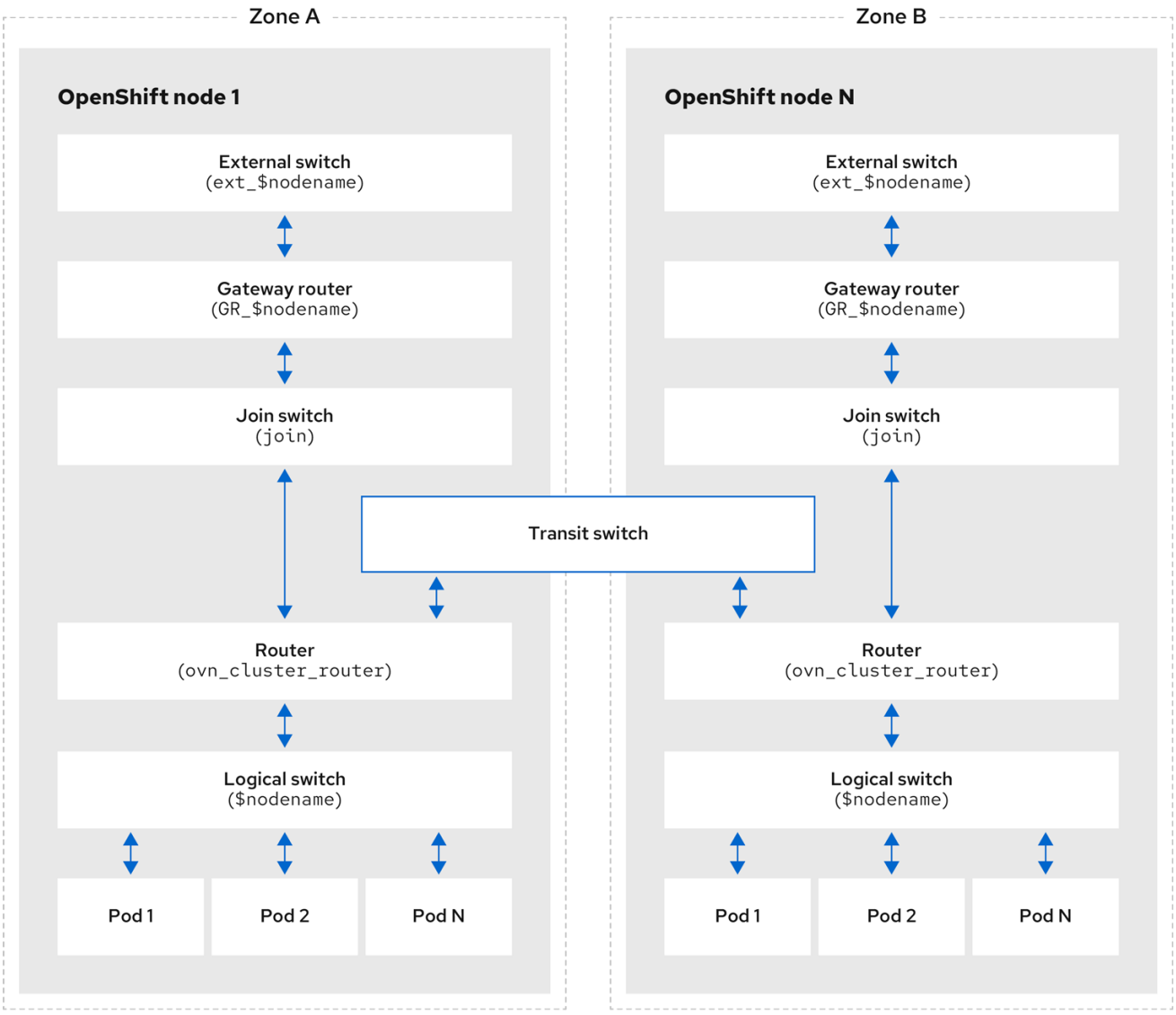
表 26.3. 检查南向数据库内容的命令行参数

参数	描述
ovn-sbctl show	从特定节点看到的南向数据库内容概述。
ovn-sbctl list Port_Binding <port>	列出特定端口的南向数据库的内容。
ovn-sbctl dump-flows	列出逻辑流。

26.2.7. OVN-Kubernetes 逻辑架构

OVN 是网络虚拟化解决方案。它创建逻辑交换机和路由器。这些交换机和路由器是互连的，以创建任何网络拓扑。当您运行 ovnkube-trace 时，日志级别设置为 2 或 5 时，OVN-Kubernetes 逻辑组件会被公开。下图显示了如何在 OpenShift Container Platform 中连接路由器和交换机。

图 26.2. OVN-Kubernetes 路由器和交换机组件



涉及数据包处理的关键组件有：

网关路由器

网关路由器有时称为 L3 网关路由器，通常在分布式路由器和物理网络之间使用。网关路由器（包括其逻辑补丁端口）绑定到物理位置（而非分布式）或机箱。此路由器上的跳接端口称为 ovn-southbound 数据库(ovn-sbdb)中的 l3gateway 端口。

分布式逻辑路由器

分布式逻辑路由器和其后面的逻辑交换机（虚拟机和容器附加）有效驻留在每个虚拟机监控程序上。

加入本地交换机

加入本地交换机用于连接分布式路由器和网关路由器。它可减少分布式路由器中所需的 IP 地址数量。

使用跳接端口的逻辑交换机

带有补丁端口的逻辑交换机用于虚拟化网络堆栈。它们通过隧道连接远程逻辑端口。

使用 localnet 端口的逻辑交换机

使用 localnet 端口的逻辑交换机用于将 OVN 连接到物理网络。它们通过将数据包桥接到使用 localnet 端口直接连接的物理 L2 片段来连接远程逻辑端口。

补丁端口

跳接端口代表逻辑交换机和逻辑路由器之间以及对等逻辑路由器之间的连接。单个连接在每个此类连接点上都有一对跳接端口。

l3gateway 端口

l3gateway 端口是 ovn-sbdb 中用于网关路由器中使用的逻辑补丁端口的端口绑定条目。它们称为 l3gateway 端口，而不是跳接端口，而只是这些端口绑定到机箱，就像网关路由器本身一样。

localnet 端口

桥接逻辑交换机上存在 localnet 端口，允许从每个 ovn-controller 实例连接到本地可访问的网络。这有助于从逻辑交换机对物理网络的直接连接建模。逻辑交换机只能附加一个 localnet 端口。

26.2.7.1. 在本地主机上安装 network-tools

在本地主机上安装 network-tools，以提供一组工具来调试 OpenShift Container Platform 集群网络问题。

流程

1. 使用以下命令将 network-tools 存储库克隆到工作站：

```
$ git clone git@github.com:openshift/network-tools.git
```

2. 更改到您刚才克隆的存储库的目录：

```
$ cd network-tools
```

3. 可选：列出所有可用的命令：

```
$ ./debug-scripts/network-tools -h
```

26.2.7.2. 运行 network-tools

通过运行 network-tools 来获取逻辑交换机和路由器的信息。

先决条件

- 已安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录集群。
- 您已在本地主机上安装了 network-tools。

流程

1. 运行以下命令列出路由器：

```
$ ./debug-scripts/network-tools ovn-db-run-command ovn-nbctl lr-list
```

输出示例

```
944a7b53-7948-4ad2-a494-82b55eeccf87 (GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99)
84bd4a4c-4b0b-4a47-b0cf-a2c32709fc53 (ovn_cluster_router)
```

2. 运行以下命令列出 localnet 端口：

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=localnet
```

输出示例

```
_uuid               : d05298f5-805b-4838-9224-1211afc2f199
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : f3c2c959-743b-4037-854d-26627902597c
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : br-ex_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : [unknown]
mirror_rules        : []
nat_addresses       : []
options             : {network_name=physnet}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 2
type                : localnet
up                  : false
virtual_parent      : []

[...]
```

3. 运行以下命令列出 l3gateway 端口：

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=l3gateway
```

输出示例

```
_uuid               : 5207a1f3-1cf3-42f1-83e9-387bbb06b03c
additional_chassis  : []
additional_encap    : []
chassis             : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath            : f3c2c959-743b-4037-854d-26627902597c
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : etor-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : ["42:01:0a:00:80:04"]
mirror_rules        : []
nat_addresses       : ["42:01:0a:00:80:04 10.0.128.4"]
options             : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772",
```

```
peer=rtoe-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port        : []
port_security      : []
requested_additional_chassis: []
requested_chassis  : []
tag                : []
tunnel_key         : 1
type               : l3gateway
up                 : true
virtual_parent     : []

_uuid              : 6088d647-84f2-43f2-b53f-c9d379042679
additional_chassis : []
additional_encap   : []
chassis            : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath           : dc9cea00-d94a-41b8-bdb0-89d42d13aa2e
encap              : []
external_ids       : {}
gateway_chassis    : []
ha_chassis_group   : []
logical_port       : jtor-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                : [router]
mirror_rules       : []
nat_addresses      : []
options            : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772",
peer=rtoj-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port        : []
port_security      : []
requested_additional_chassis: []
requested_chassis  : []
tag                : []
tunnel_key         : 2
type               : l3gateway
up                 : true
virtual_parent     : []

[...]
```

4. 运行以下命令列出跳接端口：

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=patch
```

输出示例

```
_uuid              : 785fb8b6-ee5a-4792-a415-5b1cb855dac2
additional_chassis : []
additional_encap   : []
chassis            : []
datapath           : f1ddd1cc-dc0d-43b4-90ca-12651305acec
encap              : []
external_ids       : {}
gateway_chassis    : []
ha_chassis_group   : []
logical_port       : stor-ci-ln-54932yb-72292-kd676-worker-c-rzj99
```

```
mac               : [router]
mirror_rules      : []
nat_addresses     : ["0a:58:0a:80:02:01 10.128.2.1 is_chassis_resident(\"cr-rtos-ci-ln-54932yb-72292-kd676-worker-c-rzj99\")"]
options           : {peer=rtos-ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port       : []
port_security     : []
requested_additional_chassis: []
requested_chassis  : []
tag               : []
tunnel_key        : 1
type              : patch
up                : false
virtual_parent    : []

_uuid             : c01ff587-21a5-40b4-8244-4cd0425e5d9a
additional_chassis : []
additional_encap   : []
chassis           : []
datapath          : f6795586-bf92-4f84-9222-efe4ac6a7734
encap             : []
external_ids      : {}
gateway_chassis   : []
ha_chassis_group  : []
logical_port      : rtoj-ovn_cluster_router
mac               : ["0a:58:64:40:00:01 100.64.0.1/16"]
mirror_rules      : []
nat_addresses     : []
options           : {peer=jtor-ovn_cluster_router}
parent_port       : []
port_security     : []
requested_additional_chassis: []
requested_chassis  : []
tag               : []
tunnel_key        : 1
type              : patch
up                : false
virtual_parent    : []
[...]
```

26.2.8. 其他资源

- 使用 ovnkube-trace 追踪 Openflow
- OVN 架构
- OVN-nbctl linux 手册页
- OVN-sbctl linux 手册页

26.3. OVN-KUBERNETES 故障排除

OVN-Kubernetes 具有许多内置健康检查和日志来源。按照这些部分中的说明检查集群。如果需要支持问题单，请按照 支持指南 通过 must-gather 收集更多信息。仅在支持团队要求这样做时，才使用 --gather_network_logs。

26.3.1. 使用就绪度探测监控 OVN-Kubernetes 健康状况

ovnkube-control-plane 和 ovnkube-node pod 配置有就绪度探测的容器。

先决条件

- 访问 OpenShift CLI（oc）。
- **您可以使用 cluster-admin 权限访问集群。**
- **您已安装了 jq。**

流程

1. 运行以下命令，查看 **ovnkube-node 就绪度探测的详情：**

```
$ oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node \
-o json | jq '.items[0].spec.containers[] | .name,.readinessProbe'
```

ovnkube-node pod 中的北向和南向数据库容器的就绪度探测会检查数据库和 ovnkube-controller 容器的健康状态。

ovnkube-node pod 中的 ovnkube-controller 容器有一个就绪度探测来验证 OVN-Kubernetes CNI 配置文件是否存在，这代表 pod 没有运行，或者没有准备好接受配置 pod 的请求。

2. **使用以下命令，显示命名空间的所有事件，包括探测失败：**

```
$ oc get events -n openshift-ovn-kubernetes
```

3. 仅显示特定 pod 的事件：

```
$ oc describe pod ovnkube-node-9lqfk -n openshift-ovn-kubernetes
```

4. 显示集群网络 Operator 的消息和状态：

```
$ oc get co/network -o json | jq '.status.conditions[]'
```

5. 运行以下脚本，显示 **ovnkube-node pod 中每个容器的就绪状态：**

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); do echo === $p ===; \
oc get pods -n openshift-ovn-kubernetes $p -o json | jq '.status.containerStatuses[] |
.name, .ready'; \
done
```

注意

预期的是所有容器状态都报告为 **true。就绪度探测失败，将状态设置为 false。**

其他资源

- **使用健康检查来监控应用程序的健康状态**

26.3.2. 在控制台中查看 OVN-Kubernetes 警报

Alerting UI 提供有关警报及其相关警报规则和静默的详细信息。

先决条件

- 对于您要查看指标的项目，您可以作为开发者或具有查看权限的用户访问集群。

流程 (UI)

1. 在 Administrator 视角中，选择 Observe → Alerting。在此视角中，Alerting UI 有三个主要页面，即 Alerts、Silences 和 Alerting Rules 页面。
2. 选择 Observe → Alerting → Alerting Rules 来查看 OVN-Kubernetes 警报的规则。

26.3.3. 在 CLI 中查看 OVN-Kubernetes 警报

您可以从命令行获取有关警报及其监管警报规则和静默的信息。

先决条件

- 使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。
- 您已安装了 jq。

流程

1. 运行以下命令，查看活动或触发警报：
   a. 运行以下命令设置警报管理器路由环境变量：

```
$ ALERT_MANAGER=$(oc get route alertmanager-main -n openshift-monitoring \
-o jsonpath='{@.spec.host}')
```

   b. 运行以下命令，将 $ALERT_MANAGER 替换为 Alertmanager 实例的 URL，向警报管理器路由 API 发出 curl 请求：

```
$ curl -s -k -H "Authorization: Bearer $(oc create token prometheus-k8s -n
openshift-monitoring)" https://$ALERT_MANAGER/api/v1/alerts | jq '.data[] | "\
(.labels.severity) \(.labels.alertname) \(.labels.pod) \(.labels.container) \
(.labels.endpoint) \(.labels.instance)"'
```

2. 运行以下命令来查看警报规则：

```
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -s
'http://localhost:9090/api/v1/rules' | jq '.data.groups[].rules[] |
select(((.name|contains("ovn")) or (.name|contains("OVN")) or (.name|contains("Ovn"))
or (.name|contains("North")) or (.name|contains("South"))) and .type=="alerting")'
```

26.3.4. 使用 CLI 查看 OVN-Kubernetes 日志

您可以使用 OpenShift CLI (oc)查看 ovnkube-master 和 ovnkube-node pod 中每个 pod 的日志。

先决条件

- **使用具有 cluster-admin 角色的用户访问集群。**
- 访问 OpenShift CLI（oc）。
- **您已安装了 jq。**

流程

1. 查看特定 pod 的日志：

```
$ oc logs -f <pod_name> -c <container_name> -n <namespace>
```

其中：

-f
可选：指定输出是否遵循要写到日志中的内容。
<pod_name>
指定 pod 的名称。
<container_name>
可选：指定容器的名称。当 pod 具有多个容器时，您必须指定容器名称。
<namespace>
指定 pod 运行的命名空间。

例如：

```
$ oc logs ovnkube-node-5dx44 -n openshift-ovn-kubernetes
```

```
$ oc logs -f ovnkube-node-5dx44 -c ovnkube-controller -n openshift-ovn-kubernetes
```

输出的日志文件内容。

2. 检查 **ovnkube-node** pod 中所有容器的最新条目：

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); \
do echo === $p ===; for container in $(oc get pods -n openshift-ovn-kubernetes $p \
-o json | jq -r '.status.containerStatuses[] | .name');do echo ---$container---; \
oc logs -c $container $p -n openshift-ovn-kubernetes --tail=5; done; done
```

3. **使用以下命令，查看 ovnkube-node** pod **中每个容器的最后 5 行：**

```
$ oc logs -l app=ovnkube-node -n openshift-ovn-kubernetes --all-containers --tail 5
```

26.3.5. 使用 Web 控制台查看 OVN-Kubernetes 日志

您可以在 web 控制台中查看 ovnkube-master 和 ovnkube-node pod **中每个** pod **的日志。**

先决条件

- 访问 OpenShift CLI（oc）。

流程

1. 在 OpenShift Container Platform 控制台中，导航到 Workloads → Pods，或通过您要调查的资源导航到 pod。
2. 从下拉菜单中选择 **openshift-ovn-kubernetes** 项目。
3. 点您要调查的 pod 的名称。
4. 点 Logs。默认情况下，**ovnkube-master** 显示与 **northd** 容器关联的日志。
5. 使用向下下拉菜单选择每个容器的日志。

26.3.5.1. 更改 OVN-Kubernetes 日志级别

OVN-Kubernetes 的默认日志级别为 4。要调试 OVN-Kubernetes，请将日志级别设置为 5。按照以下步骤增加 OVN-Kubernetes 的日志级别，以帮助您调试问题。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 访问 OpenShift Container Platform web 控制台。

流程

1. 运行以下命令，以获取 OVN-Kubernetes 项目中所有 pod 的详细信息：

```
$ oc get po -o wide -n openshift-ovn-kubernetes
```

输出示例

```
NAME                                READY   STATUS    RESTARTS      AGE    IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-65497d4548-9ptdr   2/2     Running   2 (128m ago)   147m
10.0.0.3     ci-ln-3njdr9b-72292-5nwkp-master-0       <none>           <none>
ovnkube-control-plane-65497d4548-j6zfk   2/2     Running   0              147m   10.0.0.5
ci-ln-3njdr9b-72292-5nwkp-master-2       <none>           <none>
ovnkube-node-5dx44                       8/8     Running   0              146m   10.0.0.3     ci-ln-
3njdr9b-72292-5nwkp-master-0       <none>           <none>
ovnkube-node-dpfn4                       8/8     Running   0              146m   10.0.0.4     ci-ln-
3njdr9b-72292-5nwkp-master-1       <none>           <none>
ovnkube-node-kwc9l                       8/8     Running   0              134m   10.0.128.2   ci-ln-
3njdr9b-72292-5nwkp-worker-a-2fjcj   <none>           <none>
ovnkube-node-mcrhl                       8/8     Running   0              134m   10.0.128.4   ci-ln-
3njdr9b-72292-5nwkp-worker-c-v9x5v   <none>           <none>
ovnkube-node-nsct4                       8/8     Running   0              146m   10.0.0.5     ci-ln-
3njdr9b-72292-5nwkp-master-2       <none>           <none>
ovnkube-node-zrj9f                       8/8     Running   0              134m   10.0.128.3   ci-ln-
3njdr9b-72292-5nwkp-worker-b-v78h7   <none>           <none>
```

2. 创建类似以下示例的 **ConfigMap** 文件，并使用文件名，如 **env-overrides.yaml** ：

ConfigMap 文件示例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: env-overrides
  namespace: openshift-ovn-kubernetes
data:
  ci-ln-3njdr9b-72292-5nwkp-master-0: | (1)
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  ci-ln-3njdr9b-72292-5nwkp-master-2: |
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  _master: | (2)
    # This sets the log level for the ovn-kubernetes master process as well as the ovn-dbchecker:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for northd, nbdb and sbdb on all masters:
    OVN_LOG_LEVEL=dbg
```

(1) 指定要设置 debug 日志级别的节点名称。

(2) 指定 _master 来设置 ovnkube-master 组件的日志级别。

3. 使用以下命令应用 ConfigMap 文件：

```
$ oc apply -n openshift-ovn-kubernetes -f env-overrides.yaml
```

输出示例

```
configmap/env-overrides.yaml created
```

4. 使用以下命令重启 ovnkube pod 以应用新的日志级别：

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-0 -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-2 -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes -l app=ovnkube-node
```

5. 要验证 'ConfigMap'file 是否已应用到特定 pod 的所有节点，请运行以下命令：

```
$ oc logs -n openshift-ovn-kubernetes --all-containers --prefix ovnkube-node-<xxxx> |
grep -E -m 10 '(Logging config:|vconsole|DBG)'
```

其中：

<XXXX>

指定上一步中 pod 的随机字符序列。

输出示例

```
[pod/ovnkube-node-2cpjc/sbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor '-
-ovn-sb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-
%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_sb_ovsdb
[pod/ovnkube-node-2cpjc/ovnkube-controller] I1012 14:39:59.984506   35767
config.go:2247] Logging config: {File: CNIFile:/var/log/ovn-kubernetes/ovn-k8s-cni-
overlay.log LibovsdbFile:/var/log/ovnkube/libovsdb.log Level:5 LogFileMaxSize:100
LogFileMaxBackups:5 LogFileMaxAge:0 ACLLoggingRateLimit:20}
[pod/ovnkube-node-2cpjc/northd] + exec ovn-northd --no-chdir -vconsole:info -
vfile:off '-vPATTERN:console:%D{%Y-%m-
%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' --pidfile /var/run/ovn/ovn-northd.pid --
n-threads=1
[pod/ovnkube-node-2cpjc/nbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor
'--ovn-nb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-
%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_nb_ovsdb
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.552Z|00002|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes,
including 1 bucket with 6 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00003|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes,
including 1 bucket with 6 nodes (64 nodes total across 64 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00004|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes,
including 1 bucket with 7 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00005|reconnect|DBG|unix:/var/run/openvswitch/db.sock:
entering BACKOFF
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00007|reconnect|DBG|unix:/var/run/openvswitch/db.sock:
entering CONNECTING
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00008|ovsdb_cs|DBG|unix:/var/run/openvswitch/db.sock:
SERVER_SCHEMA_REQUESTED -> SERVER_SCHEMA_REQUESTED at lib/ovsdb-
cs.c:423
```

6. 可选：运行以下命令来检查 ConfigMap 文件是否已应用：

```
for f in $(oc -n openshift-ovn-kubernetes get po -l 'app=ovnkube-node' --no-headers -o
custom-columns=N:.metadata.name) ; do echo "---- $f ----" ; oc -n openshift-ovn-
kubernetes exec -c ovnkube-controller $f --  pgrep -a -f  init-ovnkube-controller | grep -
P -o '^.*loglevel\s+\d' ; done
```

输出示例

```
---- ovnkube-node-2dt57 ----
60981 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-c-vmh5n.c.openshift-qe.internal --init-node xpst8-worker-c-vmh5n.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-4zznh ----
178034 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-2.c.openshift-qe.internal --init-node xpst8-master-2.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-548sx ----
77499 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-a-fjtnb.c.openshift-qe.internal --init-node xpst8-worker-a-fjtnb.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-6btrf ----
73781 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-b-p8rww.c.openshift-qe.internal --init-node xpst8-worker-b-p8rww.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-fkc9r ----
130707 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-0.c.openshift-qe.internal --init-node xpst8-master-0.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 5
---- ovnkube-node-tk9l4 ----
181328 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-1.c.openshift-qe.internal --init-node xpst8-master-1.c.openshift-qe.internal --config-file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
```

26.3.6. 检查 OVN-Kubernetes pod 网络连接

在 OpenShift Container Platform 4.10 及更新的版本中，连接检查控制器会在集群中编配连接验证检查。这包括 Kubernetes API、OpenShift API 和单个节点。连接测试的结果存储在 openshift-network-diagnostics 命名空间中的 PodNetworkConnectivity 对象中。连接测试会每分钟以并行方式执行。

先决条件

- 访问 OpenShift CLI（oc）。
- 使用具有 cluster-admin 角色的用户访问集群。
- 您已安装了 jq。

流程

1. 要列出当前的 PodNetworkConnectivityCheck 对象，请输入以下命令：

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics
```

2. 使用以下命令查看每个连接对象的最新成功：

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.successes[0]'
```

3. 使用以下命令查看每个连接对象的最新故障：

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.failures[0]'
```

4. 使用以下命令查看每个连接对象的最新中断：

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.outages[0]'
```

连接检查控制器也会将来自这些检查的指标记录到 Prometheus 中。

5. 运行以下命令来查看所有指标：

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant  http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

6. 查看源 pod 和 openshift api 服务之间的延迟，持续 5 分钟：

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant  http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

26.3.7. 其他资源

- 为红帽支持收集您的集群数据
- 连接健康检查实现
- 验证端点的网络连接

26.4. 使用 OVNKUBE-TRACE 追踪 OPENFLOW

OVN 和 OVS 流量流可以在一个名为 ovnkube-trace 的单个实用程序中模拟。ovnkube-trace 工具运行 ovn-trace、ovs-appctl ofproto/trace 和 ovn-detrace，并在单个输出中关联这些信息。

您可以从专用容器执行 ovnkube-trace 二进制文件。对于 OpenShift Container Platform 4.7 后的版本，您还可以将该二进制文件复制到本地主机中，并从该主机执行它。

26.4.1. 在本地主机上安装 ovnkube-trace

ovnkube-trace 工具跟踪在 OVN-Kubernetes 驱动的 OpenShift Container Platform 集群中点之间的任意 UDP 或 TCP 流量的数据包模拟。将 ovnkube-trace 二进制文件复制到本地主机，使其可以针对集群运行。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

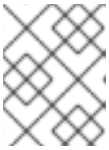
流程

1. 使用以下命令创建 pod 变量：

```
$  POD=$(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-control-plane -o
name | head -1 | awk -F '/' '{print $NF}')
```

2. 在本地主机上运行以下命令，从 ovnkube-control-plane pod 复制二进制文件：

```
$ oc cp -n openshift-ovn-kubernetes $POD:/usr/bin/ovnkube-trace -c ovnkube-cluster-manager ovnkube-trace
```



注意

如果您使用 Red Hat Enterprise Linux (RHEL) 8 运行 ovnkube-trace 工具，您必须将文件 /usr/lib/rhel8/ovnkube-trace 复制到本地主机。

3. 运行以下命令，使 ovnkube-trace 可执行：

```
$ chmod +x ovnkube-trace
```

4. 运行以下命令，显示 ovnkube-trace 可用的选项：

```
$ ./ovnkube-trace -help
```

预期输出

```
Usage of ./ovnkube-trace:
  -addr-family string
    Address family (ip4 or ip6) to be used for tracing (default "ip4")
  -dst string
    dest: destination pod name
  -dst-ip string
    destination IP address (meant for tests to external targets)
  -dst-namespace string
    k8s namespace of dest pod (default "default")
  -dst-port string
    dst-port: destination port (default "80")
  -kubeconfig string
    absolute path to the kubeconfig file
  -loglevel string
    loglevel: klog level (default "0")
  -ovn-config-namespace string
    namespace used by ovn-config itself
  -service string
    service: destination service name
  -skip-detrace
    skip ovn-detrace command
  -src string
    src: source pod name
  -src-namespace string
    k8s namespace of source pod (default "default")
  -tcp
    use tcp transport protocol
  -udp
    use udp transport protocol
```

支持的命令行参数是熟悉的 Kubernetes 构造，如命名空间、Pod、服务，因此您不需要查找 MAC 地址、目标节点的 IP 地址或 ICMP 类型。

日志级别为：

- 0（最小输出）
- 2（显示 trace 命令结果的详细输出）
- 5（调试输出）

26.4.2. 运行 ovnkube-trace

运行 ovn-trace 以模拟 OVN 逻辑网络内的数据包转发。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您已在本地主机上安装了 ovnkube-trace

示例：测试 DNS 解析是否适用于部署的 pod

本例演示了如何从部署的 pod 测试 DNS 解析到集群中运行的核心 DNS pod。

流程

1. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --
labels="app=web" --expose --port=80
```

2. 列出在 openshift-dns 命名空间中运行的 pod：

```
oc get pods -n openshift-dns
```

输出示例

```
NAME                READY   STATUS    RESTARTS   AGE
dns-default-8s42x     2/2     Running   0          5h8m
dns-default-mdw6r     2/2     Running   0          4h58m
dns-default-p8t5h     2/2     Running   0          4h58m
dns-default-rl6nk     2/2     Running   0          5h8m
dns-default-xbgqx     2/2     Running   0          5h8m
dns-default-zv8f6     2/2     Running   0          4h58m
node-resolver-62jjb   1/1     Running   0          5h8m
node-resolver-8z4cj   1/1     Running   0          4h59m
node-resolver-bq244   1/1     Running   0          5h8m
node-resolver-hc58n   1/1     Running   0          4h59m
node-resolver-lm6z4   1/1     Running   0          5h8m
node-resolver-zfx5k   1/1     Running   0          5h
```

3. 运行以下 ovnkube-trace 命令来验证 DNS 解析是否正常工作：

```
$ ./ovnkube-trace \
```

```
-src-namespace default \ 1
-src web \ 2
-dst-namespace openshift-dns \ 3
-dst dns-default-p8t5h \ 4
-udp -dst-port 53 \ 5
-loglevel 0 6
```

1. 源 pod 的命名空间
2. 源 pod 名称
3. 目标 pod 的命名空间
4. 目标 pod 名称
5. 使用 udp 传输协议。端口 53 是 DNS 服务使用的端口。
6. 将日志级别设置为 0 (0 为 minimal，5 为 debug)

当 src&dst pod 位于同一节点上的输出示例：

```
ovn-trace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-trace destination pod to source pod indicates success from dns-default-p8t5h to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-default-p8t5h
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-p8t5h to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-detrace destination pod to source pod indicates success from dns-default-p8t5h to web
```

当 src&dst pod 位于不同节点上的输出示例：

```
ovn-trace source pod to destination pod indicates success from web to dns-default-8s42x
ovn-trace (remote) source pod to destination pod indicates success from web to dns-default-8s42x
ovn-trace destination pod to source pod indicates success from dns-default-8s42x to web
ovn-trace (remote) destination pod to source pod indicates success from dns-default-8s42x to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-default-8s42x
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-8s42x to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-8s42x
ovn-detrace destination pod to source pod indicates success from dns-default-8s42x to web
```

ouput 表示从部署的 pod 到 DNS 端口的成功，也表示它已成功返回其他方向。因此，如果我的 Web pod 想要从核心 DNS 进行 dns 解析，则 UDP 端口 53 上支持双向流量。

例如，如果无法正常工作，并且您想要获取 ovn-trace、proto/trace 和 ovn-detrace 的 ovs-appctl，以及更多故障排除类型信息，将日志级别增加到 2，然后再次运行命令，如下所示：

```
$ ./ovnkube-trace \
  -src-namespace default \
  -src web \
  -dst-namespace openshift-dns \
  -dst dns-default-467qw \
  -udp -dst-port 53 \
  -loglevel 2
```

这个日志级别的输出太大，无法在此处列出。在故障情况下，此命令的输出显示哪个流丢弃了该流量。例如，可以在不允许该流量的集群中配置 egress 或 ingress 网络策略。

示例：使用 debug 输出来验证配置的默认拒绝

本例演示了如何使用入口默认拒绝策略阻断流量的 debug 输出来识别。

流程

1. 创建以下 YAML，以定义 deny-by-default 策略，以拒绝所有命名空间中的所有 pod 的入口流量。将 YAML 保存到 deny-by-default.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: default
spec:
  podSelector: {}
  ingress: []
```

2. 输入以下命令应用策略：

```
$ oc apply -f deny-by-default.yaml
```

输出示例

```
networkpolicy.networking.k8s.io/deny-by-default created
```

3. 输入以下命令在 default 命名空间中启动 web 服务：

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --
labels="app=web" --expose --port=80
```

4. 运行以下命令来创建 prod 命名空间：

```
$ oc create namespace prod
```

5. 运行以下命令来标记 prod 命名空间：

```
$ oc label namespace/prod purpose=production
```

6. 运行以下命令，在 prod 命名空间中部署 alpine 镜像并启动 shell：

```
$ oc run test-6459 --namespace=prod --rm -i -t --image=alpine -- sh
```

7. 打开另一个终端会话。

8. 在这个新终端会话中，运行 ovn-trace 以验证在命名空间 prod 中运行的源 pod test-6459 与在 default 命名空间中的目标 pod 之间的通信失败：

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 0
```

输出示例

```
ovn-trace source pod to destination pod indicates failure from test-6459 to web
```

9. 运行以下命令，将日志级别增加到 2 以公开失败的原因：

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 2
```

输出示例

```
...
------------------------------------------------
 3. ls_out_acl_hint (northd.c:7454): !ct.new && ct.est && !ct.rpl && ct_mark.blocked ==
0, priority 4, uuid 12efc456
    reg0[8] = 1;
    reg0[10] = 1;
    next;
 5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 0, priority 500, uuid 69372c5d
    reg8[30..31] = 1;
    next(4);
 5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 1, priority 500, uuid 2fa0af89
    reg8[30..31] = 2;
    next(4);
 4. ls_out_acl_eval (northd.c:7691): reg8[30..31] == 2 && reg0[10] == 1 && (outport ==
@a16982411286042166782_ingressDefaultDeny), priority 2000, uuid 447d0dab
    reg8[17] = 1;
```

```
    ct_commit { ct_mark.blocked = 1; }; (1)
    next;
...
```

(1) 因为默认的 deny 策略被到位，所以入口流量会被阻止

10. 创建一个策略，允许来自特定命名空间中所有 pod 的流量，其标签为 purpose=production。将 YAML 保存到 web-allow-prod.yaml 文件中：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production
```

11. 输入以下命令应用策略：

```
$ oc apply -f web-allow-prod.yaml
```

12. 输入以下命令运行 ovnkube-trace 来验证现在允许的流量：

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 0
```

预期输出

```
ovn-trace source pod to destination pod indicates success from test-6459 to web
ovn-trace destination pod to source pod indicates success from web to test-6459
ovs-appctl ofproto/trace source pod to destination pod indicates success from test-
6459 to web
ovs-appctl ofproto/trace destination pod to source pod indicates success from web to
test-6459
ovn-detrace source pod to destination pod indicates success from test-6459 to web
ovn-detrace destination pod to source pod indicates success from web to test-6459
```

13. 在在第 6 步中打开的 shell 中运行以下命令，以将 nginx 连接到 web-server：

```
wget -qO- --timeout=2 http://web.default
```

预期输出

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

26.4.3. 其他资源

- 使用 ovnkube-trace 工具追踪 Openflow
- ovnkube-trace

26.5. 从 OPENSHIFT SDN 网络插件迁移

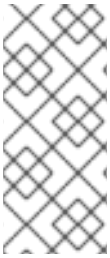
作为集群管理员，您可以使用 *离线* 迁移方法或有限的实时迁移方法从 OpenShift SDN 网络插件迁移到 OVN-Kubernetes 网络插件。

要了解更多有关 OVN-Kubernetes 的信息，请参阅关于 OVN-Kubernetes 网络插件。

26.5.1. 离线迁移到 OVN-Kubernetes 网络插件概述

离线迁移方法是一个手动过程，其中包括一些停机时间，在此期间集群无法访问。此方法主要用于自我管理的 OpenShift Container Platform 部署。

虽然提供了一个回滚过程，但离线迁移通常被认为是一个单向过程。



注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

以下小节提供了有关离线迁移方法的更多信息。

26.5.1.1. 使用离线迁移方法时支持的平台

下表提供了有关离线迁移类型的支持的平台信息。

表 26.4. 离线迁移方法支持的平台

平台	离线迁移
裸机硬件 (IPI 和 UPI)	✓
Amazon Web Services (AWS) (IPI 和 UPI)	✓
Google Cloud Platform (GCP) (IPI 和 UPI)	✓
IBM Cloud® (IPI and UPI)	✓
Microsoft Azure (IPI 和 UPI)	✓
Red Hat OpenStack Platform (RHOSP) (IPI 和 UPI)	✓
VMware vSphere (IPI 和 UPI)	✓
AliCloud (IPI 和 UPI)	✓
Nutanix (IPI 和 UPI)	✓

26.5.1.2. 离线迁移到 OVN-Kubernetes 网络插件的注意事项

如果您在 OpenShift Container Platform 集群中有超过 150 个节点，请创建一个支持问题单，供您迁移到 OVN-Kubernetes 网络插件。

迁移过程中不会保留分配给节点的子网以及分配给各个 pod 的 IP 地址。

虽然 OVN-Kubernetes 网络插件实现 OpenShift SDN 网络插件中存在的许多功能，但配置并不相同。

- **如果您的集群使用以下 OpenShift SDN 网络插件功能，您必须在 OVN-Kubernetes 网络插件中手动配置相同的功能：**
  - **命名空间隔离**
  - **出口路由器 pod**

- OVN-Kubernetes 是 OpenShift Container Platform 4.14 及之后的版本中的默认网络供应商，在内部使用以下 IP 地址范围：**100.64.0.0/16**, **169.254.169.0/29**, **100.88.0.0/16**, **fd98::/64**, **fd69::/125**, 和 **fd97::/64**。如果您的集群使用 OVN-Kubernetes，请不要在集群或基础架构中的任何其他 CIDR 定义中包含这些 IP 地址范围。

以下小节重点介绍了上述功能在 OVN-Kubernetes 和 OpenShift SDN 网络插件中的配置差异。

命名空间隔离
OVN-Kubernetes 仅支持网络策略隔离模式。

重要

对于使用在多租户或子网隔离模式下配置的 OpenShift SDN 的集群，您仍然可以迁移到 OVN-Kubernetes 网络插件。请注意，在迁移操作后，多租户隔离模式会被丢弃，因此您必须手动配置网络策略，以便为 Pod 和服务达到相同的项目级别的隔离。

出口 IP 地址
OpenShift SDN 支持两种不同的 Egress IP 模式：

- 在*自动分配*方法中，给节点分配一个出口 IP 地址范围。
- 在*手动分配*方法中，给节点分配包含一个或多个出口 IP 地址的列表。

迁移过程支持迁移使用自动分配模式的 Egress IP 配置。

下表中描述了在 OVN-Kubernetes 和 OpenShift SDN 配置出口 IP 地址的不同：

表 26.5. 出口 IP 地址配置的不同

OVN-Kubernetes	OpenShift SDN
- 创建 **EgressIPs** 对象 - 在一个 **Node** 对象上添加注解	- 对 **NetNamespace** 对象进行补丁 - 对 **HostSubnet** 对象进行补丁

有关在 OVN-Kubernetes 中使用出口 IP 地址的更多信息，请参阅"配置出口 IP 地址"。

出口网络策略
下表中描述在 OVN-Kubernetes 和 OpenShift SDN 间配置出口网络策略（也称为出口防火墙）的不同之处：

表 26.6. 出口网络策略配置的不同

OVN-Kubernetes	OpenShift SDN
- 在命名空间中创建 **EgressFirewall** 对象	- 在命名空间中创建一个 **EgressNetworkPolicy** 对象

注意

由于 EgressFirewall 对象的名称只能设置为 default，在迁移后，所有迁移的 EgressNetworkPolicy 对象都会命名为 default，而无论在 OpenShift SDN 下的名称是什么。

如果您随后回滚到 OpenShift SDN，则所有 EgressNetworkPolicy 对象都会命名为 default，因为之前的名称已丢失。

有关在 OVN-Kubernetes 中使用出口防火墙的更多信息，请参阅"配置项目出口防火墙"。

出口路由器 pod
OVN-Kubernetes 支持重定向模式的出口路由器 pod。OVN-Kubernetes 不支持 HTTP 代理模式或 DNS 代理模式的出口路由器 pod。

使用 Cluster Network Operator 部署出口路由器时，您无法指定节点选择器来控制用于托管出口路由器 pod 的节点。

多播
下表中描述了在 OVN-Kubernetes 和 OpenShift SDN 上启用多播流量的区别：

表 26.7. 多播配置的不同

OVN-Kubernetes	OpenShift SDN
• 在 **Namespace** 对象上添加注解	• 在 **NetNamespace** 对象中添加注解

有关在 OVN-Kubernetes 中使用多播的更多信息，请参阅"启用项目多播"。

网络策略
OVN-Kubernetes 在 networking.k8s.io/v1 API 组中完全支持 KubernetesNetworkPolicy API。从 OpenShift SDN 进行迁移时，网络策略不需要更改。

26.5.1.3. 离线迁移过程如何工作

下表对迁移过程进行了概述，它分为操作中的用户发起的步骤，以及在响应过程中迁移过程要执行的操作。

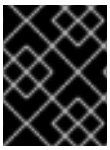
表 26.8. 从 OpenShift SDN 离线迁移到 OVN-Kubernetes

用户发起的步骤	迁移操作
将名为 **cluster** 的 **Network.operator.openshift.io** 自定义资源（CR）的 **migration** 字段设置为 **OVNKubernetes**。在设置值之前，请确保 **migration** 项为 **null**。	Cluster Network Operator (CNO) 相应地更新名为 **cluster** 的 **Network.config.openshift.io** CR 的状态。 Machine Config Operator（MCO） 将更新发布到 OVN-Kubernetes 所需的 systemd 配置 ; MCO 默认更新每个池的单一机器，从而导致迁移总时间随着集群大小而增加。

用户发起的步骤	迁移操作
更新 **Network.config.openshift.io** CR 的 **networkType** 字段。	CNO 执行以下操作： • 销毁 OpenShift SDN control plane pod。 • 部署 OVN-Kubernetes control plane pod。 • 更新 Multus 守护进程集和配置映射对象，以反映新的网络插件。
重新引导集群中的每个节点。	Cluster 当节点重启时，集群会为 OVN-Kubernetes 集群网络上的 pod 分配 IP 地址。

26.5.1.4. 使用离线迁移方法迁移到 OVN-Kubernetes 网络插件

作为集群管理员，您可以将集群的网络插件更改为 OVN-Kubernetes。在迁移过程中，您必须重新引导集群中的每个节点。



重要

在进行迁移时，集群不可用，工作负载可能会中断。仅在服务中断可以接受时才执行迁移。

先决条件

- **在网络策略隔离模式中使用 OpenShift SDN CNI 网络插件配置集群。**
- **安装 OpenShift CLI（oc）。**
- **使用具有 cluster-admin 角色的用户访问集群。**
- **etcd 数据库的最新备份可用。**
- **可根据每个节点手动触发重新引导。**
- **集群处于已知良好状态，没有任何错误。**
- **在迁移到 OVN-Kubernetes 之前，必须有一条安全组规则，以允许所有云平台上所有节点的 UDP 数据包在端口 6081 上。**

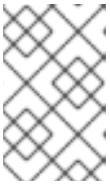
流程

1. **要备份集群网络的配置，请输入以下命令：**

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. 要为迁移准备所有节点，请输入以下命令在 Cluster Network Operator 配置对象上设置 migration 字段：

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": { "networkType": "OVNKubernetes" } } }'
```



注意

此步骤不会立即部署 OVN-Kubernetes。相反，指定 migration 字段会触发 Machine Config Operator（MCO）将新机器配置应用到集群中的所有节点，以准备 OVN-Kubernetes 部署。

3. 可选：您可以禁用将几个 OpenShift SDN 功能自动迁移到 OVN-Kubernetes 等效功能：

   - 出口 IP
   - 出口防火墙
   - 多播

   要为之前记录的 OpenShift SDN 功能禁用配置自动迁移，请指定以下键：

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{
    "spec": {
      "migration": {
        "networkType": "OVNKubernetes",
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

   其中：

   bool：指定是否启用功能的迁移。默认值是 true。

4. 可选： 您可以自定义 OVN-Kubernetes 的以下设置，以满足您的网络基础架构要求：

   - 最大传输单元 (MTU)。在为这个可选步骤自定义 MTU 前请考虑以下几点：
     - 如果您使用默认 MTU，并且要在迁移期间保留默认 MTU，则可以忽略这一步。
     - 如果您使用自定义 MTU，并且要在迁移过程中保留自定义 MTU，则必须在此步骤中声明自定义 MTU 值。
     - 如果要在迁移过程中更改 MTU 值，此步骤将无法正常工作。相反，您必须首先按照"选择集群 MTU"的说明进行操作。然后，您可以通过执行此步骤并声明自定义 MTU 值来保留自定义 MTU 值。

注意

OpenShift-SDN 和 OVN-Kubernetes 具有不同的覆盖（overlay）开销。MTU 值应遵循在 "MTU 值选择" 页面中找到的准则来选择。

- Geneve（Generic Network Virtualization Encapsulation）覆盖网络端口
- OVN-Kubernetes IPv4 内部子网
- OVN-Kubernetes IPv6 内部子网

要自定义之前记录的设置之一，请输入以下命令。如果您不需要更改默认值，请从补丁中省略该键。

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "mtu":<mtu>,
          "genevePort":<port>,
          "v4InternalSubnet":"<ipv4_subnet>",
          "v6InternalSubnet":"<ipv6_subnet>"
    }}}}'
```

其中：

mtu

Geneve 覆盖网络的 MTU。这个值通常是自动配置的；但是，如果集群中的节点没有都使用相同的 MTU，那么您必须将此值明确设置为比最小节点 MTU 的值小 100。

port

Geneve 覆盖网络的 UDP 端口。如果没有指定值，则默认为 6081。端口不能与 OpenShift SDN 使用的 VXLAN 端口相同。VXLAN 端口的默认值为 4789。

ipv4_subnet

OVN-Kubernetes 内部使用的 IPv4 地址范围。您必须确保 IP 地址范围没有与 OpenShift Container Platform 安装使用的任何其他子网重叠。IP 地址范围必须大于可添加到集群的最大节点数。默认值为 100.64.0.0/16。

ipv6_subnet

OVN-Kubernetes 内部使用的 IPv6 地址范围。您必须确保 IP 地址范围没有与 OpenShift Container Platform 安装使用的任何其他子网重叠。IP 地址范围必须大于可添加到集群的最大节点数。默认值为 fd98::/48。

更新 mtu 字段的 patch 命令示例

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "mtu":1200
    }}}}'
```

5. 当 MCO 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get mcp
```

成功更新的节点具有以下状态：UPDATED=true、UPDATING=false、DEGRADED=false。

注意

默认情况下，MCO 会一次在一个池中更新一个机器，从而导致迁移总时间随着集群大小的增加而增加。

6. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

```
$ oc describe node | egrep "hostname|machineconfig"
```

   输出示例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

   验证以下语句是否正确：

   - machineconfiguration.openshift.io/state 字段的值为 Done。
   - machineconfiguration.openshift.io/currentConfig 字段的值等于 machineconfiguration.openshift.io/desiredConfig 字段的值。

   b. 要确认机器配置正确，请输入以下命令：

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

   这里的 <config_name> 是 machineconfiguration.openshift.io/currentConfig 字段中机器配置的名称。

   机器配置必须包括以下对 systemd 配置的更新：

```
ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
```

   c. 如果节点一直处于 NotReady 状态，检查机器配置守护进程 pod 日志并解决所有错误。

      i. 运行以下命令列出 pod：

```
$ oc get pod -n openshift-machine-config-operator
```

      输出示例

      -

```
NAME                                READY   STATUS    RESTARTS   AGE
machine-config-controller-75f756f89d-sjp8b   1/1     Running   0          37m
machine-config-daemon-5cf4b         2/2     Running   0          43h
machine-config-daemon-7wzcd         2/2     Running   0          43h
machine-config-daemon-fc946         2/2     Running   0          43h
machine-config-daemon-g2v28         2/2     Running   0          43h
machine-config-daemon-gcl4f         2/2     Running   0          43h
machine-config-daemon-l5tnv         2/2     Running   0          43h
machine-config-operator-79d9c55d5-hth92     1/1     Running   0          37m
machine-config-server-bsc8h         1/1     Running   0          43h
machine-config-server-hklrm         1/1     Running   0          43h
machine-config-server-k9rtx         1/1     Running   0          43h
```

配置守护进程 pod 的名称使用以下格式：machine-config-daemon-<seq>。<seq> 值是一个随机的五个字符的字母数字序列。

ii. 使用以下命令，输出在上一个输出中显示的第一个机器配置守护进程 pod 的 pod 日志：

```
$ oc logs <pod> -n openshift-machine-config-operator
```

其中 pod 是机器配置守护进程 pod 的名称。

iii. 解决上一命令输出中显示的日志中的任何错误。

7. 要启动迁移，请使用以下命令配置 OVN-Kubernetes 网络插件：

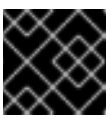
- 要指定网络供应商而不更改集群网络 IP 地址块，请输入以下命令：

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

- 要指定不同的集群网络 IP 地址块，请输入以下命令：

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{
    "spec": {
      "clusterNetwork": [
        {
          "cidr": "<cidr>",
          "hostPrefix": <prefix>
        }
      ],
      "networkType": "OVNKubernetes"
    }
  }'
```

其中 cidr 是 CIDR 块，prefix 是集群中每个节点的 CIDR 块的分片。您不能使用任何与 10064.0.0/16 CIDR 块重叠的 CIDR 块，因为 OVN-Kubernetes 网络供应商在内部使用此块。



重要

您无法在迁移过程中更改服务网络地址块。

8. 在继续执行后续步骤前，验证 Multus 守护进程集的 rollout 是否已完成：

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus pod 的名称采用 multus-<xxxxx> 的形式，其中 <xxxxx>是由字母组成的随机序列。pod 可能需要一些时间才能重启。

输出示例

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

9. 要完成更改网络插件，请重新引导集群中的每个节点。您可以使用以下方法之一重新引导集群中的节点：

- 使用 oc rsh 命令，您可以使用类似如下的 bash 脚本：

```
#!/bin/bash
readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o wide| grep daemon|awk '{print $1" "$7}')"

for i in "${POD_NODES[@]}"
do
  read -r POD NODE <<< "$i"
  until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
    do
      echo "cannot reboot node $NODE, retry" && sleep 3
    done
done
```

- 通过 ssh 命令，您可以使用类似如下的 bash 脚本：该脚本假设您已将 sudo 配置为不提示输入密码。

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}')
do
   echo "reboot node $ip"
   ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

10. 确认迁移成功完成：

a. 要确认网络插件是 OVN-Kubernetes，请输入以下命令。status.networkType 的值必须是 OVNKubernetes。

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

b. 要确认集群节点处于 Ready 状态，请输入以下命令：

```
$ oc get nodes
```

c. 要确认您的 pod 不在错误状态，请输入以下命令：

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

如果节点上的 pod 处于错误状态，请重新引导该节点。

d. 要确认所有集群 Operator 没有处于异常状态，请输入以下命令：

```
$ oc get co
```

每个集群 Operator 的状态必须是： AVAILABLE="True"、PROGRESSING="False" 和 DEGRADED="False"。如果 Cluster Operator 不可用或降级，请检查集群 Operator 的日志以了解更多信息。

11. 只有在迁移成功且集群处于良好状态时完成以下步骤：

a. 要从 CNO 配置对象中删除迁移配置，请输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

b. 要删除 OpenShift SDN 网络供应商的自定义配置，请输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "openshiftSDNConfig": null } } }'
```

c. 要删除 OpenShift SDN 网络供应商命名空间，请输入以下命令：

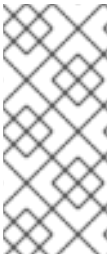
```
$ oc delete namespace openshift-sdn
```

26.5.2. 有限实时迁移到 OVN-Kubernetes 网络插件概述

有限的实时迁移方法是在不中断服务的情况下将 OpenShift SDN 网络插件及其网络配置、连接和相关资源迁移到 OVN-Kubernetes 网络插件的过程。它可用于 OpenShift Container Platform、Red Hat OpenShift Dedicated、Red Hat OpenShift Service on AWS 和 Azure Red Hat OpenShift 部署类型。它不适用于 HyperShift 部署类型。这个迁移方法对于需要持续服务可用性的部署类型非常重要，并具有以下优点：

- 持续服务可用性
- 最小化停机时间
- 自动节点重新引导
- 从 OpenShift SDN 网络插件无缝过渡到 OVN-Kubernetes 网络插件

虽然提供了一个回滚过程，但有限的实时迁移通常被认为是一个单向过程。



注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

以下小节提供了有关有限实时迁移方法的更多信息。

26.5.2.1. 使用有限实时迁移方法时支持的平台

下表提供了有关有限实时迁移类型的支持的平台信息。

表 26.9. 有限实时迁移方法支持的平台

平台	有限实时迁移
裸机硬件 (IPI 和 UPI)	✓
Amazon Web Services (AWS) (IPI 和 UPI)	✓
Google Cloud Platform (GCP) (IPI 和 UPI)	✓
IBM Cloud® (IPI and UPI)	✓
Microsoft Azure (IPI 和 UPI)	✓
Red Hat OpenStack Platform (RHOSP) (IPI 和 UPI)	✓
VMware vSphere (IPI 和 UPI)	✓
AliCloud (IPI 和 UPI)	✓
Nutanix (IPI 和 UPI)	✓

26.5.2.2. 有限实时迁移到 OVN-Kubernetes 网络插件的注意事项

在使用有限实时迁移方法到 OVN-Kubernetes 网络插件之前，集群管理员应考虑以下信息：

- **启用 OpenShift SDN 多租户模式的集群不支持有限的实时迁移步骤。**
- **出口路由器 pod 会阻止有限的实时迁移过程。在开始有限的实时迁移过程前，必须删除它们。**
- **在有限的实时迁移过程中，会临时禁用多播、多播、出口 IP 地址和出口防火墙。在有限实时迁移过程完成后，可以从 OpenShift SDN 迁移到 OVN-Kubernetes。**
- **迁移旨在成为单向过程。但是，对于希望回滚到 OpenShift-SDN 的用户，从 OpenShift-SDN 迁移到 OVN-Kubernetes 必须已成功。用户可以按照以下步骤从 OVN-Kubernetes 网络插件迁移到 OpenShift SDN 网络插件。**
- **HyperShift 集群中不支持有限的实时迁移。**

- OpenShift SDN 不支持 IPsec。迁移后，集群管理员可以启用 IPsec。
- OpenShift SDN 不支持 IPv6。迁移后，集群管理员可以启用双栈。
- 集群 MTU 是 pod 接口的 MTU 值。它始终小于硬件 MTU，以考虑集群网络覆盖开销。OVN-Kubernetes 的开销为 100 字节，OpenShift SDN 是 50 字节。
  在有限的实时迁移过程中，OVN-Kubernetes 和 OpenShift SDN 并行运行。OVN-Kubernetes 管理某些节点的集群网络，而 OpenShift SDN 管理其他用户的集群网络。为确保跨 CNI 流量保持正常工作，Cluster Network Operator 会更新可路由的 MTU，以确保两个 CNI 共享相同的覆盖 MTU。因此，在迁移完成后，集群 MTU 小于 50 字节。
- 安装后无法更改 OVN-Kubernetes 的一些参数。只有启动有限的实时迁移前才能设置以下参数：
  - **InternalTransitSwitchSubnet**
  - **internalJoinSubnet**
- OVN-Kubernetes 会保留 **100.64.0.0/16** 和 **100.88.0.0/16** IP 地址范围。如果 OpenShift SDN 被配置为使用这些 IP 地址范围之一，您必须在启动有限的实时迁移前将它们进行补丁为使用不同的 IP 地址范围。
  - **100.64.0.0/16**.默认情况下，此 IP 地址范围用于 OVN-Kubernetes 的 **internalJoinSubnet** 参数。如果这个 IP 地址范围已经在使用中，输入以下命令将其更新至不同的范围，例如 **100.63.0.0/16** ：

    ```
    $ oc patch network.operator.openshift.io cluster --type='merge' -p='{"spec":
    {"defaultNetwork":{"ovnKubernetesConfig":{"ipv4":{"internalJoinSubnet":
    "100.63.0.0/16"}}}}}'
    ```

  - **100.88.0.0/16**.默认情况下，此 IP 地址范围用于 OVN-Kubernetes 的 **internalTransSwitchSubnet** 参数。如果这个 IP 地址范围已经在使用中，输入以下命令将其更新至不同的范围，例如 **100.99.0.0/16** ：

    ```
    $ oc patch network.operator.openshift.io cluster --type='merge' -p='{"spec":
    {"defaultNetwork":{"ovnKubernetesConfig":{"ipv4":
    {"internalTransitSwitchSubnet": "100.99.0.0/16"}}}}}'
    ```

- 在大多数情况下，有限的实时迁移独立于 Multus CNI 插件创建的 pod 的二级接口。但是，如果这些二级接口在主机的默认网络接口控制器 (NIC) 上设置，例如，使用 MACVLAN、IPVLAN、SR-IOV 或桥接接口作为控制节点，则 OVN-Kubernetes 可能会遇到故障。在继续有限的实时迁移前，用户应删除此配置。
- 当主机中存在多个 NIC 时，且默认路由不在具有 Kubernetes NodeIP 的接口上，则必须使用离线迁移。
- 在启动有限实时迁移前，必须删除 **openshift-sdn** 命名空间中的所有 **DaemonSet** 对象（不受 Cluster Network Operator (CNO) 管理。这些非受管守护进程集可能会导致迁移状态在未正确处理时保持不完整。

26.5.2.3. 实时迁移过程如何工作

下表总结了实时迁移过程，具体是流程中用户发起的步骤与迁移脚本在响应中执行的操作之间的部分。

表 26.10. 从 OpenShiftSDN 实时迁移到 OVNKubernetes

用户发起的步骤	迁移操作
通过将 **networkType** 从 **OpenShiftSDN** 改为 **OVNKubernetes** 来修补集群级别的网络配置。	Cluster Network Operator (CNO) ● 在 **network.operator** 自定义资源 (CR) 中设置迁移相关字段，并等待可路由 MTU 应用到所有节点。 ● 对 **network.operator** CR 进行补丁，将 OVN-Kubernetes 的迁移模式设置为 **Live**，并以迁移模式部署 OpenShift SDN 网络插件。 ● 部署启用了混合覆盖的 OVN-Kubernetes，确保不会发生任何操作条件。 ● 等待 OVN-Kubernetes 部署并更新 **network.config** CR 状态中的条件。 ● 触发 Machine Config Operator (MCO)，将新机器配置应用到每个机器配置池，其中包括节点封锁、排空和重启。 ● OVN-Kubernetes 将节点添加到适当的区中，并使用 OVN-Kubernetes 作为默认 CNI 插件重新创建 pod。 ● 从 network.operator CR 中删除与迁移相关的字段，并执行清理操作，如删除 OpenShift SDN 资源，并使用必要的配置以正常模式重新部署 OVN-Kubernetes。 ● 等待 OVN-Kubernetes 重新部署并更新 **network.config** CR 中的状态条件，以指示迁移完成。如果您的迁移被阻止，请参阅"检查有限的实时迁移指标"以了解有关对问题进行故障排除的信息。

26.5.2.4. 使用有限的实时迁移方法迁移到 OVN-Kubernetes 网络插件

以下流程检查出口路由器资源，并使用有限的实时迁移方法从 OpenShift SDN 网络插件迁移到 OVN-Kubernetes 网络插件。

先决条件

- **在网络策略隔离模式下，使用 OpenShift SDN CNI 网络插件配置集群。**
- **已安装 OpenShift CLI(oc)。**
- **您可以使用具有 cluster-admin 角色的用户访问集群。**
- **您已创建了 etcd 数据库的最新备份。**
- **集群处于已知良好状态，且没有任何错误。**

- 在迁移到 OVN-Kubernetes 之前，必须有一条安全组规则，以允许所有云平台上所有节点的 UDP 数据包在端口 6081 上。
- 在开始有限的实时迁移前，集群管理员必须删除所有出口路由器 pod。如需有关出口路由器 pod 的更多信息，请参阅"以重定向模式部署出口路由器 pod"。
- 您已参阅本文档的"正在考虑的有限实时迁移"部分。

流程

1. 在启动有限的实时迁移前，您必须检查任何出口路由器 pod。如果在执行有限的实时迁移时集群中有一个出口路由器 pod，Network Operator 会阻断迁移并返回以下错误：

```
The cluster configuration is invalid (network type live migration is not supported for pods with `pod.network.openshift.io/assign-macvlan` annotation. Please remove all egress router pods). Use `oc edit network.config.openshift.io cluster` to fix.
```

   - 输入以下命令查找集群中的出口路由器 pod：

```
$ oc get pods --all-namespaces -o json | jq '.items[] |
select(.metadata.annotations."pod.network.openshift.io/assign-macvlan" ==
"true") | {name: .metadata.name, namespace: .metadata.namespace}'
```

     输出示例

```
{
  "name": "egress-multi",
  "namespace": "egress-router-project"
}
```

   - 另外，您还可以查询 OpenShift Container Platform Web 控制台的指标，或使用 oc CLI 检查出口路由器 pod。如需更多信息，请参阅"检查有限的实时迁移指标"。

2. 输入以下命令删除出口路由器 pod:

```
$ oc delete pod <egress_pod_name> -n <egress_router_project>
```

3. 输入以下命令来修补集群级别的网络配置，并启动从 OpenShift SDN 到 OVN-Kubernetes 的迁移：

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch '{"metadata":
{"annotations":{"network.openshift.io/network-type-migration":""}},"spec":
{"networkType":"OVNKubernetes"}}'
```

   运行这些命令后，迁移过程开始。在此过程中，Machine Config Operator 会重启集群中的节点两次。在集群升级时，迁移大约需要两倍。

4. 可选：您可以输入以下命令来确保迁移过程已完成，并检查 network.config 的状态：

```
$ oc get network.config.openshift.io cluster -o jsonpath='{.status.networkType}'
```

```
$ oc get network.config cluster -o=jsonpath='{.status.conditions}' | jq .
```

您可以检查有限的实时迁移指标，以进行故障排除。如需更多信息，请参阅"检查有限的实时迁移指标"。

5. 只有在迁移成功且集群处于良好状态时完成以下步骤：

   a. 要从 network.config 自定义资源中删除迁移配置，请输入以下命令：

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": null } }'
   ```

   b. 要删除 OpenShift SDN 网络供应商的自定义配置，请输入以下命令：

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "defaultNetwork": { "openshiftSDNConfig": null } } }'
   ```

   c. 要删除 OpenShift SDN 网络供应商命名空间，请输入以下命令：

   ```
   $ oc delete namespace openshift-sdn
   ```

26.5.2.5. 检查有限的实时迁移指标

可用于监控有限实时迁移的进度。可以在 OpenShift Container Platform Web 控制台中或使用 oc CLI 查看指标。

先决条件

- 您已启动了一个有限实时迁移到 OVN-Kubernetes。

流程

1. 在 OpenShift Container Platform Web 控制台中查看有限的实时迁移指标：

   a. 点 Observe → Metrics。

   b. 在 Expression 框中，键入 openshift_network 并点 openshift_network_operator_live_migration_procedure 选项。

2. 使用 oc CLI 查看指标：

   a. 输入以下命令为 openshift-monitoring 命名空间中的 prometheus-k8s 服务帐户生成令牌：

   ```
   $ oc create token prometheus-k8s -n openshift-monitoring
   ```

   输出示例

   ```
   eyJhbGciOiJSUzI1NiIsImtpZCI6IlZiSUtwcIcwbEJ2VW9We...
   ```

   b. 输入以下命令请求有关 openshift_network_operator_live_migration_condition 指标的信息：

   ```
   $ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -k -H
   "Authorization: <eyJhbGciOiJSUzI1NiIsImtpZCI6IlZiSUtwcIcwbEJ2VW9We...>"
   "https://<openshift_API_endpoint>" --data-urlencode
   "query=openshift_network_operator_live_migration_condition" | jq`
   ```

输出示例

```
"status": "success",
 "data": {
  "resultType": "vector",
  "result": [
   {
    "metric": {
     "__name__": "openshift_network_operator_live_migration_condition",
     "container": "network-operator",
     "endpoint": "metrics",
     "instance": "10.0.83.62:9104",
     "job": "metrics",
     "namespace": "openshift-network-operator",
     "pod": "network-operator-6c87754bc6-c8qld",
     "prometheus": "openshift-monitoring/k8s",
     "service": "metrics",
     "type": "NetworkTypeMigrationInProgress"
    },
    "value": [
     1717653579.587,
     "1"
    ]
   },
...
```

"Information about limited live migration metrics" 中的表显示可用的指标，以及 openshift_network_operator_live_migration_procedure 表达式填充的标签值。使用这些信息来监控进度或对迁移进行故障排除。

26.5.2.5.1. 关于有限实时迁移指标的信息

下表显示了可用的指标以及从 openshift_network_operator_live_migration_procedure 表达式填充的标签值。使用这些信息来监控进度或对迁移进行故障排除。

表 26.11. 有限的实时迁移指标

指标	标签值

指标	标签值
openshift_network_operator_live_migration_blocked: Prometheus 量表向量指标。包含恒定的 **1** 值的指标，带有 CNI 有限实时迁移的原因可能没有启动。当 CNI 有限实时迁移通过注解 **Network** 自定义资源时，此指标可用。 除非有限实时迁移被阻止，否则不会发布此指标。	标签值列表包括以下内容 • **UnsupportedCNI**: 无法迁移到不支持的目标 CNI。从 OpenShift SDN 迁移时，有效的 CNI 是 **OVNKubernetes**。 • **UnsupportedHyperShiftCluster** ：在 HCP 集群中不支持有限实时迁移。 • **UnsupportedSDNNetworkIsolationMode**: OpenShift SDN 配置了一个不支持的网络隔离模式 **Multitenant**。在执行有限的实时迁移前，迁移到受支持的网络隔离模式。 • **UnsupportedMACVLANInterface** ：删除出口路由器或包含 pod 注解 **pod.network.openshift.io/assign-macvlan** 的任何 pod。使用以下命令查找 offending pod 的命名空间或 pod 名称： **oc get pods -Ao=jsonpath='{range .items[?(@.metadata.annotations.pod\.network\.openshift\.io/assign-macvlan=="")]} {@.metadata.namespace}{"\t"} {@.metadata.name}{"\n"}'**。
openshift_network_operator_live_migration_condition: 代表 CNI 限制实时迁移的每个条件类型的状态的指标。为 **network.config** 定义了一组状态条件类型，以支持 CNI 有限实时迁移的可观察性。 **1** 表示条件状态为 **true**。**0** 代表 **false**。**-1** 代表未知。当 CNI 有限实时迁移通过注解 **Network** 自定义资源时，此指标可用。 只有通过向 **Network** CR 集群添加相关注解来触发有限实时迁移时，此指标才可用，否则不会被发布。如果 Network CR 集群中没有以下条件类型，则会清除指标及其标签。	标签值列表包括以下内容 • **NetworkTypeMigrationInProgress** • **NetworkTypeMigrationTargetCNIAvailable** • **NetworkTypeMigrationTargetCNIInUse** • **NetworkTypeMigrationOriginalCNIPurged** • **NetworkTypeMigrationMTUReady**

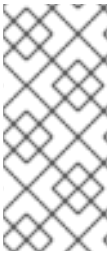
26.5.3. 其他资源

- Red Hat OpenShift Network Calculator
- OVN-Kubernetes 网络插件的配置参数
- 备份 etcd

- 关于网络策略
- 更改集群 MTU
- MTU 值选择
- OVN-Kubernetes 功能
  - 配置出口 IP 地址
  - 为项目配置出口防火墙
  - 为项目启用多播
- OpenShift SDN 功能
  - 为项目配置出口 IP
  - 为项目配置出口防火墙
  - 为项目启用多播
  - 以重定向模式部署出口路由器 pod
- Network [operator.openshift.io/v1]

26.6. 回滚到 OPENSHIFT SDN 网络供应商

作为集群管理员，您可以使用 *离线* 迁移方法或有限的实时迁移方法从 OVN-Kubernetes 网络插件回滚到 OpenShift SDN 网络插件。这只能在迁移到 OVN-Kubernetes 网络插件后完成。



注意

- 如果您使用离线迁移方法从 OVN-Kubernetes 网络插件迁移到 OpenShift SDN 网络插件，您应该使用离线迁移回滚方法。
- 如果您使用有限的实时迁移方法从 OVN-Kubernetes 网络插件迁移到 OpenShift SDN 网络插件，则应使用有限的实时迁移回滚方法。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

26.6.1. 使用离线迁移方法回滚到 OpenShift SDN 网络插件

作为集群管理员，您可以使用离线迁移方法回滚到 OpenShift SDN Container Network Interface (CNI) 网络插件。在迁移过程中，您必须手动重新引导集群中的每个节点。使用离线迁移方法时，集群会存在一些停机时间。

重要

在启动回滚前，您必须等待 OpenShift SDN 到 OVN-Kubernetes 网络插件的迁移过程成功。

如果需要回滚到 OpenShift SDN，下表描述了这个过程。

表 26.12. 执行到 OpenShift SDN 的回滚

用户发起的步骤	迁移操作
挂起 MCO 以确保它不会中断迁移。	MCO 停止。
将名为 **cluster** 的 **Network.operator.openshift.io** 自定义资源(CR)的 **migration** 字段设置为 **OpenShiftSDN**。在设置值之前，请确保 **migration** 项为 **null**。	CNO 相应地更新名为 **cluster** 的 **Network.config.openshift.io** CR 的状态。
更新 **networkType** 字段。	CNO 执行以下操作： ● 销毁 OVN-Kubernetes control plane pod。 ● 部署 OpenShift SDN control plane pod。 ● 更新 Multus 对象以反映新的网络插件。
重新引导集群中的每个节点。	Cluster 当节点重启时，集群会为 OpenShift-SDN 网络上的 pod 分配 IP 地址。
在集群重启中的所有节点后启用 MCO。	MCO 将更新发布到 OpenShift SDN 所需的 systemd 配置；MCO 默认更新每个池的单一机器，因此迁移总时间随着集群的大小而增加。

先决条件

- **已安装 OpenShift CLI (oc)。**
- **可以使用具有 cluster-admin 角色的用户访问集群。**
- **集群安装在使用 OVN-Kubernetes 网络插件配置的基础架构上。**
- **etcd 数据库的最新备份可用。**
- **可以为每个节点触发手动重新引导。**

- 集群处于已知良好状态，没有任何错误。

流程

1. 停止由 Machine Config Operator（MCO）管理的所有机器配置池：
   - 停止 master 配置池：

     ```
     $ oc patch MachineConfigPool master --type='merge' --patch \
       '{ "spec": { "paused": true } }'
     ```

   - 停止 worker 机器配置池：

     ```
     $ oc patch MachineConfigPool worker --type='merge' --patch \
       '{ "spec":{ "paused": true } }'
     ```

2. 要准备迁移，请输入以下命令将 migration 字段设置为 null ：

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": null } }'
   ```

3. 要开始迁移，请输入以下命令将网络插件设置为 OpenShift SDN：

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "migration": { "networkType": "OpenShiftSDN" } } }'

   $ oc patch Network.config.openshift.io cluster --type='merge' \
     --patch '{ "spec": { "networkType": "OpenShiftSDN" } }'
   ```

4. 可选：您可以将多个 OVN-Kubernetes 功能的自动迁移到 OpenShift SDN 等效功能：
   - 出口 IP
   - 出口防火墙
   - 多播

   要为之前记录的 OpenShift SDN 功能禁用配置自动迁移，请指定以下键：

   ```
   $ oc patch Network.operator.openshift.io cluster --type='merge' \
     --patch '{
       "spec": {
         "migration": {
           "networkType": "OpenShiftSDN",
           "features": {
             "egressIP": <bool>,
             "egressFirewall": <bool>,
             "multicast": <bool>
           }
         }
       }
     }'
   ```

   其中：

bool：指定是否启用功能的迁移。默认值是 true。

5. 可选：您可以自定义 OpenShift SDN 的以下设置，以满足您的网络基础架构的要求：

   - 最大传输单元（MTU）
   - VXLAN 端口

   要自定义之前记录的设置或其中的一个设置，进行自定义并输入以下命令。如果您不需要更改默认值，请从补丁中省略该键。

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "openshiftSDNConfig":{
          "mtu":<mtu>,
          "vxlanPort":<port>
    }}}}'
```

   mtu

   VXLAN 覆盖网络的 MTU。这个值通常是自动配置的；但是，如果集群中的节点没有都使用相同的 MTU，那么您必须将此值明确设置为比最小节点 MTU 的值小 50。

   port

   VXLAN 覆盖网络的 UDP 端口。如果没有指定值，则默认为 4789。端口不能与 OVN-Kubernetes 使用的生成端口相同。Geneve 端口的默认值为 6081。

   patch 命令示例

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "openshiftSDNConfig":{
          "mtu":1200
    }}}}'
```

6. 重新引导集群中的每个节点。您可以使用以下方法之一重新引导集群中的节点：

   - 使用 oc rsh 命令，您可以使用类似如下的 bash 脚本：

```
#!/bin/bash
readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator
-o wide| grep daemon|awk '{print $1" "$7}')"

for i in "${POD_NODES[@]}"
do
  read -r POD NODE <<< "$i"
  until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs
shutdown -r +1
    do
      echo "cannot reboot node $NODE, retry" && sleep 3
    done
done
```

- 通过 ssh 命令，您可以使用类似如下的 bash 脚本：该脚本假设您已将 sudo 配置为不提示输入密码。

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

7. 等待 Multus 守护进程集的 rollout 完成。运行以下命令查看您的 rollout 状态：

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus pod 的名称采用 multus-<xxxxx> 的形式，其中 <xxxxx>是由字母组成的随机序列。pod 可能需要一些时间才能重启。

输出示例

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been
updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

8. 重新引导集群中的节点并推出 multus pod 后，通过运行以下命令启动所有机器配置池：

- 启动 master 配置池：

```
$ oc patch MachineConfigPool master --type='merge' --patch \
  '{ "spec": { "paused": false } }'
```

- 启动 worker 配置池：

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
  '{ "spec": { "paused": false } }'
```

当 MCO 更新每个配置池中的机器时，它会重新引导每个节点。

默认情况下，MCO 会在一个时间段内为每个池更新一台机器，因此迁移完成所需要的时间会随集群大小的增加而增加。

9. 确认主机上新机器配置的状态：

a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

```
$ oc describe node | egrep "hostname|machineconfig"
```

输出示例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
```

```
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

验证以下语句是否正确：

- machineconfiguration.openshift.io/state 字段的值为 Done。
- machineconfiguration.openshift.io/currentConfig 字段的值等于 machineconfiguration.openshift.io/desiredConfig 字段的值。

b. 要确认机器配置正确，请输入以下命令：

```
$ oc get machineconfig <config_name> -o yaml
```

这里的 <config_name> 是 machineconfiguration.openshift.io/currentConfig 字段中机器配置的名称。

10. 确认迁移成功完成：

a. 要确认网络插件是 OpenShift SDN，请输入以下命令。status.networkType 的值必须是 OpenShiftSDN。

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

b. 要确认集群节点处于 Ready 状态，请输入以下命令：

```
$ oc get nodes
```

c. 如果节点一直处于 NotReady 状态，检查机器配置守护进程 pod 日志并解决所有错误。

i. 运行以下命令列出 pod：

```
$ oc get pod -n openshift-machine-config-operator
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
machine-config-controller-75f756f89d-sjp8b   1/1     Running   0          37m
machine-config-daemon-5cf4b         2/2     Running   0          43h
machine-config-daemon-7wzcd         2/2     Running   0          43h
machine-config-daemon-fc946         2/2     Running   0          43h
machine-config-daemon-g2v28         2/2     Running   0          43h
machine-config-daemon-gcl4f         2/2     Running   0          43h
machine-config-daemon-l5tnv         2/2     Running   0          43h
machine-config-operator-79d9c55d5-hth92     1/1     Running   0          37m
machine-config-server-bsc8h         1/1     Running   0          43h
machine-config-server-hklrm         1/1     Running   0          43h
machine-config-server-k9rtx         1/1     Running   0          43h
```

配置守护进程 pod 的名称使用以下格式：machine-config-daemon-<seq>。<seq> 值是一个随机的五个字符的字母数字序列。

ii. 要显示上一输出中显示的每个机器配置守护进程 pod 的 pod 日志，请输入以下命令：

```
$ oc logs <pod> -n openshift-machine-config-operator
```

其中 pod 是机器配置守护进程 pod 的名称。

iii. 解决上一命令输出中显示的日志中的任何错误。

d. 要确认您的 pod 不在错误状态，请输入以下命令：

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

如果节点上的 pod 处于错误状态，请重新引导该节点。

11. 只有在迁移成功且集群处于良好状态时完成以下步骤：

a. 要从 Cluster Network Operator 配置对象中删除迁移配置，请输入以下命令：

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "migration": null } }'
```

b. 要删除 OVN-Kubernetes 配置，请输入以下命令：

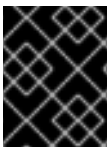
```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "ovnKubernetesConfig":null } } }'
```

c. 要删除 OVN-Kubernetes 网络供应商命名空间，请输入以下命令：

```
$ oc delete namespace openshift-ovn-kubernetes
```

26.6.2. 使用有限的实时迁移方法回滚到 OpenShift SDN 网络插件

作为集群管理员，您可以使用有限的实时迁移方法回滚到 OpenShift SDN Container Network Interface (CNI) 网络插件。使用此方法迁移过程中，节点会自动重新引导，并且对集群的服务不会中断。



重要

在启动回滚前，您必须等待 OpenShift SDN 到 OVN-Kubernetes 网络插件的迁移过程成功。

如果需要回滚到 OpenShift SDN，下表描述了这个过程。

表 26.13. 执行到 OpenShift SDN 的回滚

用户发起的步骤	迁移操作

用户发起的步骤	迁移操作
通过将 **networkType** 从 **OVNKubernetes** 改为 **OpenShiftSDN** 来修补集群级别的网络配置。	Cluster Network Operator (CNO) 执行以下操作： ● 在 **network.operator** 自定义资源 (CR) 中设置与迁移相关的字段，并等待 Machine Config Operator (MCO) 将可路由 MTU 应用到所有节点。 ● 对 **network.operator** CR 进行补丁，为 OpenShiftSDN 将 migration 模式设置为 **Live**，并在迁移模式中部署 OpenShiftSDN 网络插件。 ● 在启用了混合覆盖的情况下部署 OVN-Kubernetes。 ● 等待两个 CNI 插件被部署并更新 **network.config** CR 状态中的条件。 ● 触发 MCO 将新机器配置应用到每个机器配置池，其中包括节点封锁、排空和重启。 ● 从 **network.operator** CR 中删除与迁移相关的字段，并执行清理操作，如删除 OpenShift SDN 资源，并使用必要的配置以正常模式重新部署 OVN-Kubernetes。 ● 等待 OpenShiftSDN 重新部署并更新 **network.config** CR 中的状态条件，以指示迁移完成。

先决条件

- **已安装 OpenShift CLI (oc)。**
- **可以使用具有 cluster-admin 角色的用户访问集群。**
- **集群安装在使用 OVN-Kubernetes 网络插件配置的基础架构上。**
- **etcd 数据库的最新备份可用。**
- **可以为每个节点触发手动重新引导。**
- **集群处于已知良好状态，没有任何错误。**

流程

1. **要启动回滚到 OpenShift SDN，请输入以下命令：**

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch '{"metadata":
{"annotations":{"network.openshift.io/network-type-migration":""}},"spec":
{"networkType":"OpenShiftSDN"}}'
```

2. 要监控迁移的进度，请输入以下命令：

```
$ watch -n1 'oc get network.config/cluster -o json | jq ".status.conditions[]|\"\\(.type) \\
(.status) \\(.reason) \\(.message)\""  -r | column --table --table-columns
NAME,STATUS,REASON,MESSAGE --table-columns-limit 4; echo; oc get mcp -o wide;
echo; oc get node -o "custom-
columns=NAME:metadata.name,STATE:metadata.annotations.machineconfiguration\\.
openshift\\.io/state,DESIRED:metadata.annotations.machineconfiguration\\.openshift\\.
io/desiredConfig,CURRENT:metadata.annotations.machineconfiguration\\.openshift\\.i
o/currentConfig,REASON:metadata.annotations.machineconfiguration\\.openshift\\.io/r
eason"'
```

该命令每秒打印以下信息：

- network.config.openshift.io/cluster 对象的状态的条件，报告迁移的进度。
- 与 machine-config-operator 资源相关的不同节点的状态，包括它们是否正在升级或已升级，以及它们的当前和所需的配置。

3. 在回滚步骤完成后，输入以下命令从 network.config 自定义资源中删除 network.openshift.io/network-type-migration= 注解：

```
$ oc annotate network.config cluster network.openshift.io/network-type-migration-
```

26.7. 转换为 IPV4/IPV6 双栈网络

作为集群管理员，您可以将 IPv4 单栈集群转换为支持 IPv4 和 IPv6 地址系列的双网络集群网络。转换为双栈网络后，新的和现有 pod 启用了双栈网络。

在裸机、IBM Power®、IBM Z® 基础架构、单节点 OpenShift 和 VMware vSphere 上置备的集群支持双栈网络。

重要

当使用需要 IPv6 的双栈网络时，您无法使用 IPv4 映射 IPv6 地址，如 ::FFFF:198.51.100.1。

26.7.1. 转换为双栈集群网络

作为集群管理员，您可以将单堆栈集群网络转换为双栈集群网络。转换为双栈网络后，新的和现有 pod 启用了双栈网络。

将单堆栈集群网络转换为双栈集群网络包括创建补丁并将其应用到集群的网络和子网。

注意

更改 clusterNetwork、serviceNetwork、apiServerInternalIPs 和 ingressIP 对象的每个补丁操作都会触发重启集群。更改 MachineNetworks 对象不会导致重启集群。

如果您需要将 API 和 Ingress 服务的 IPv6 虚拟 IP (VIP) 添加到现有的双栈配置集群中，您只需要修补集群的基础架构，而不是集群的网络。

重要

如果您已经将集群升级到 OpenShift Container Platform 4.16 或更高版本，且您需要将单堆栈集群网络转换为双栈集群网络，则必须从 install-config.yaml 文件中为 API 和 Ingress 服务指定现有的 IPv4 machineNetwork 网络配置。此配置可确保 IPv4 流量与默认网关在同一网络接口中。

带有为 machineNetwork 网络添加 IPv4 地址块的 YAML 配置文件示例

```
- op: add
  path: /spec/platformSpec/baremetal/machineNetworks/- 1
  value: 192.168.1.0/24
  # ...
```

1 确保为机器操作的 machineNetwork 网络指定地址块。您必须为机器网络选择 API 和 Ingress IP 地址。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 集群使用 OVN-Kubernetes 网络插件。
- 集群节点具有 IPv6 地址。
- 您已根据基础架构配置了启用了 IPv6 的路由器。

流程

1. 要为集群和服务网络指定 IPv6 地址块，请创建一个类似以下示例的 YAML 配置补丁文件：

```
- op: add
  path: /spec/clusterNetwork/-
  value: 1
    cidr: fd01::/48
    hostPrefix: 64
- op: add
  path: /spec/serviceNetwork/-
  value: fd02::/112 2
```

1 使用 cidr 和 hostPrefix 字段指定对象。主机前缀必须为 64 或更高。IPv6 无类别域间路由 (CIDR) 前缀必须足够大，以适应指定的主机前缀。

2 指定一个带有 112 前缀的 IPv6 CIDR。Kubernetes 仅使用最低 16 位。对于前缀112，IP 地址从 112 分配给 128 位。

2. 在 CLI 中输入以下命令来修补集群网络配置：

```
$ oc patch network.config.openshift.io cluster \ 1
  --type='json' --patch-file <file>.yaml
```

1 其中 file 指定您创建的 YAML 文件的名称。

输出示例

```
network.config.openshift.io/cluster patched
```

3. 要为 API 和 Ingress 服务指定 IPv6 VIP，请创建一个类似以下示例的 YAML 配置补丁文件：

```
- op: add
  path: /spec/platformSpec/baremetal/machineNetworks/- 1
  value: fd2e:6f44:5dd8::/64
- op: add
  path: /spec/platformSpec/baremetal/apiServerInternalIPs/- 2
  value: fd2e:6f44:5dd8::4
- op: add
  path: /spec/platformSpec/baremetal/ingressIPs/-
  value: fd2e:6f44:5dd8::5
```

1 确保为机器操作的 machineNetwork 网络指定地址块。您必须为机器网络选择 API 和 Ingress IP 地址。

2 确保根据您的平台指定每个文件路径。该示例展示了裸机平台上的文件路径。

4. 在 CLI 中输入以下命令来修补集群基础架构：

```
$ oc patch infrastructure cluster \ 1
  --type='json' --patch-file <file>.yaml
```

1 其中 file 指定您创建的 YAML 文件的名称。

输出示例

```
infrastructure/cluster patched
```

验证

1. 在 CLI 中输入以下命令来显示集群网络配置：

```
$ oc describe network
```

2. 检查集群网络配置是否识别您在 YAML 文件中指定的 IPv6 地址块，以验证在网络配置中成功安装补丁。

输出示例

```
# ...
Status:
  Cluster Network:
    Cidr:               10.128.0.0/14
    Host Prefix:        23
```

```
    Cidr:              fd01::/48
    Host Prefix:       64
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    172.30.0.0/16
    fd02::/112
# ...
```

3. 在 CLI 中输入以下命令来显示集群基础架构配置：

```
$ oc describe network
```

4. 检查基础架构是否识别您在 YAML 文件中指定的 IPv6 地址块，以验证在集群基础架构上安装补丁是否成功。

输出示例

```
# ...
spec:
# ...
  platformSpec:
    baremetal:
      apiServerInternalIPs:
      - 192.168.123.5
      - fd2e:6f44:5dd8::4
      ingressIPs:
      - 192.168.123.10
      - fd2e:6f44:5dd8::5
status:
# ...
  platformStatus:
    baremetal:
      apiServerInternalIP: 192.168.123.5
      apiServerInternalIPs:
      - 192.168.123.5
      - fd2e:6f44:5dd8::4
      ingressIP: 192.168.123.10
      ingressIPs:
      - 192.168.123.10
      - fd2e:6f44:5dd8::5
# ...
```

26.7.2. 转换为单堆栈集群网络

作为集群管理员，您可以将双栈集群网络转换为单堆栈集群网络。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 集群使用 OVN-Kubernetes 网络插件。

- 集群节点具有 IPv6 地址。
- 您已启用了双栈网络。

流程

1. 运行以下命令来编辑 networks.config.openshift.io 自定义资源 (CR)：

```
$ oc edit networks.config.openshift.io
```

2. 删除在前面的步骤中添加到 cidr 和 hostPrefix 字段中的 IPv6 具体配置。

26.8. 配置 OVN-KUBERNETES 内部 IP 地址子网

作为集群管理员，您可以更改 OVN-Kubernetes 网络插件用于加入和传输子网的 IP 地址范围。

26.8.1. 配置 OVN-Kubernetes 加入子网

您可以更改 OVN-Kubernetes 使用的 join 子网，以避免与环境中已存在的子网冲突。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 确保集群使用 OVN-Kubernetes 网络插件。

流程

1. 要更改 OVN-Kubernetes 加入子网，请输入以下命令：

```
$ oc patch network.operator.openshift.io cluster --type='merge' \
  -p='{"spec":{"defaultNetwork":{"ovnKubernetesConfig":
    {"ipv4":{"internalJoinSubnet": "<join_subnet>"},
    "ipv6":{"internalJoinSubnet": "<join_subnet>"}}}}}'
```

其中：

<join_subnet>

指定 OVN-Kubernetes 内部使用的 IP 地址子网。子网必须大于集群中的节点数量，且必须足够大，以适应集群中的每个节点一个 IP 地址。此子网不能与 OpenShift Container Platform 或主机本身使用的任何其他子网重叠。IPv4 的默认值为 100.64.0.0/16，IPv6 的默认值是 fd98::/64。

输出示例

```
network.operator.openshift.io/cluster patched
```

验证

- 要确认配置处于活跃状态，请输入以下命令：

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.defaultNetwork}"
```

可能需要 30 分钟才能使此更改生效。

输出示例

```
{
  "ovnKubernetesConfig": {
    "ipv4": {
      "internalJoinSubnet": "100.64.1.0/16"
    },
  },
  "type": "OVNKubernetes"
}
```

26.8.2. 配置 OVN-Kubernetes 传输子网

您可以更改 OVN-Kubernetes 使用的传输子网，以避免与环境中已存在的子网冲突。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 确保集群使用 OVN-Kubernetes 网络插件。

流程

1. 要更改 OVN-Kubernetes 传输子网，请输入以下命令：

```
$ oc patch network.operator.openshift.io cluster --type='merge' \
  -p='{"spec":{"defaultNetwork":{"ovnKubernetesConfig":
    {"ipv4":{"internalTransitSwitchSubnet": "<transit_subnet>"},
    "ipv6":{"internalTransitSwitchSubnet": "<transit_subnet>"}}}}}'
```

其中：

<transit_subnet>

为分布式传输交换机指定一个 IP 地址子网，以启用 east-west 流量。此子网不能与 OVN-Kubernetes 或主机本身使用的任何其他子网重叠。IPv4 的默认值为 100.88.0.0/16，IPv6 的默认值是 fd97::/64。

输出示例

```
network.operator.openshift.io/cluster patched
```

验证

- 要确认配置处于活跃状态，请输入以下命令：

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.defaultNetwork}"
```

可能需要 30 分钟才能使此更改生效。

输出示例

```
{
  "ovnKubernetesConfig": {
    "ipv4": {
      "internalTransitSwitchSubnet": "100.88.1.0/16"
    },
  },
  "type": "OVNKubernetes"
}
```

26.9. 在默认网络中配置外部网关

作为集群管理员，您可以在默认网络上配置外部网关。

此功能提供以下优点：

- 根据每个命名空间对出口流量进行精细控制
- 静态和动态外部网关 IP 地址的灵活配置
- 支持 IPv4 和 IPv6 地址系列

26.9.1. 先决条件

- 集群使用 OVN-Kubernetes 网络插件。
- 您的基础架构配置为路由来自二级外部网关的流量。

26.9.2. OpenShift Container Platform 如何决定外部网关 IP 地址

您可以使用 **k8s.ovn.org** API 组中的 **AdminPolicyBasedExternalRoute** 自定义资源 (CR) 配置二级外部网关。CR 支持静态和动态方法来指定外部网关的 IP 地址。

AdminPolicyBasedExternalRoute CR 目标的每个命名空间不能被任何其他 **AdminPolicyBasedExternalRoute** CR 选择。命名空间不能有并发二级外部网关。

对策略的更改在控制器中被隔离。如果策略应用失败，对其他策略的更改不会触发其他策略的重试。策略只会重新评估，在对策略本身或与策略相关的对象（如目标命名空间、Pod 网关或命名空间从动态跃点托管它们）时应用更改可能出现的差异。

静态分配

您可以直接指定 IP 地址。

动态分配

您可以间接指定 IP 地址，带有命名空间和 pod 选择器，以及可选的网络附加定义。

- 如果提供了网络附加定义的名称，则使用网络附加定义的外部网关 IP 地址。

- **如果没有提供网络附加定义的名称，则使用 pod 本身的外部网关 IP 地址。但是，只有在 pod 配置为将 hostNetwork 设置为 true 时，此方法才能正常工作。**

26.9.3. AdminPolicyBasedExternalRoute 对象配置

您可以使用以下属性来定义集群范围的 AdminPolicyBasedExternalRoute 对象。命名空间一次只能由一个 AdminPolicyBasedExternalRoute CR 选择。

表 26.14. **AdminPolicyBasedExternalRoute** 对象

字段	类型	描述
metadata.name	**string**	指定 **AdminPolicyBasedExternalRoute** 对象的名称。
spec.from	**string**	指定路由策略应用到的命名空间选择器。外部流量只支持 **namespaceSelector**。例如： `from:` `  namespaceSelector:` `  matchLabels:` `  kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059` 命名空间只能由一个 **AdminPolicyBasedExternalRoute** CR 为目标。如果命名空间由多个 **AdminPolicyBasedExternalRoute** CR 选择，则第二个和后续 CR 中针对同一命名空间会出现一个 **failed** 错误状态。要应用更新，您必须将策略本身或相关的对象更改为策略，如目标命名空间、Pod 网关或命名空间，以便从动态跃点托管它们，以便策略重新评估并要应用您的更改。
spec.nextHops	**object**	指定数据包转发到的目的地。需要是 **static** 和 **dynamic** 中的一个或这两个。您必须至少定义一个下一个跃点。

表 26.15. **nextHops** 对象

字段	类型	描述
static	**数组**	指定静态 IP 地址的数组。
dynamic	**数组**	指定与配置了网络附加定义的 pod 对应的 pod 选择器，用作外部网关目标。

表 26.16. **nextHops.static** 对象

字段	类型	描述
ip	**string**	指定下一个目的地跃点的 IPv4 或 IPv6 地址。
bfdEnabled	**布尔值**	可选：指定网络是否支持 Bi-Directional Forwarding Detection (BFD)。默认值为 **false**。

表 26.17. **nextHops.dynamic** 对象

字段	类型	描述
podSelector	**string**	指定一个 [set-based] (https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#set-based-requirement) 标签选择器来过滤与此网络配置匹配的命名空间中的 pod。
namespaceSelector	**string**	指定一个 **set-based** 选择器，用于过滤 **podSelector** 应用到的命名空间。必须为此字段指定一个值。
bfdEnabled	**布尔值**	可选：指定网络是否支持 Bi-Directional Forwarding Detection (BFD)。默认值为 **false**。
networkAttachmentName	**string**	可选：指定网络附加定义的名称。名称必须与与 pod 关联的逻辑网络列表匹配。如果没有指定此字段，则使用 pod 的主机网络。但是，pod 必须配置为主机网络 pod，才能使用主机网络。

26.9.3.1. 二级外部网关配置示例

在以下示例中，AdminPolicyBasedExternalRoute 对象使用 kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059 标签在命名空间中将两个静态 IP 地址配置为 pod 的外部网关。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: default-route-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059
  nextHops:
    static:
    - ip: "172.18.0.8"
    - ip: "172.18.0.9"
```

在以下示例中，AdminPolicyBasedExternalRoute 对象配置一个动态外部网关。用于外部网关的 IP 地址派生自与所选 pod 关联的额外网络附加。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: shadow-traffic-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        externalTraffic: ""
  nextHops:
    dynamic:
    - podSelector:
        matchLabels:
          gatewayPod: ""
      namespaceSelector:
        matchLabels:
          shadowTraffic: ""
      networkAttachmentName: shadow-gateway
    - podSelector:
        matchLabels:
          gigabyteGW: ""
      namespaceSelector:
        matchLabels:
          gatewayNamespace: ""
      networkAttachmentName: gateway
```

在以下示例中，AdminPolicyBasedExternalRoute 对象同时配置静态和动态外部网关。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: multi-hop-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        trafficType: "egress"
  nextHops:
    static:
    - ip: "172.18.0.8"
    - ip: "172.18.0.9"
    dynamic:
    - podSelector:
        matchLabels:
          gatewayPod: ""
      namespaceSelector:
        matchLabels:
          egressTraffic: ""
      networkAttachmentName: gigabyte
```

26.9.4. 配置二级外部网关

您可以为集群中的命名空间在默认网络中配置外部网关。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 创建包含 AdminPolicyBasedExternalRoute 对象的 YAML 文件。
2. 要创建基于 admin 策略的外部路由，请输入以下命令：

```
$ oc create -f <file>.yaml
```

其中：

<file>
指定您在上一步中创建的 YAML 文件的名称。

输出示例

```
adminpolicybasedexternalroute.k8s.ovn.org/default-route-policy created
```

3. 要确认已创建了基于 admin 策略的外部路由，请输入以下命令：

```
$ oc describe apbexternalroute <name> | tail -n 6
```

其中：

<name>
指定 AdminPolicyBasedExternalRoute 对象的名称。

输出示例

```
Status:
  Last Transition Time:  2023-04-24T15:09:01Z
  Messages:
  Configured external gateway IPs: 172.18.0.8
  Status:  Success
Events:  <none>
```

26.9.5. 其他资源

- 有关额外网络附加的更多信息，请参阅了解多个网络

26.10. 配置出口 IP 地址

作为集群管理员，您可以配置 OVN-Kubernetes Container Network Interface (CNI) 网络插件，为命名空间分配一个或多个出口 IP 地址，或分配给命名空间中的特定 pod。

26.10.1. 出口 IP 地址架构设计和实施

OpenShift Container Platform 出口 IP 地址功能可确保来自一个或多个命名空间中的一个或多个 pod 的流量具有集群网络之外的服务具有一致的源 IP 地址。

例如，您可能有一个 pod 定期查询托管在集群外服务器上的数据库。要强制对服务器进行访问要求，将数据包过滤设备配置为只允许来自特定 IP 地址的流量。为确保您可以可靠地允许从该特定 pod 访问服务器，您可以为向服务器发出请求的 pod 配置特定的出口 IP 地址。

分配给命名空间的出口 IP 地址与用来向特定目的地发送流量的出口路由器不同。

在一些集群配置中，应用程序 Pod 和入口路由器 pod 在同一个节点上运行。如果您在这种情况下为应用程序项目配置出口 IP 地址，当您向应用程序项目发送请求时，不会使用 IP 地址。

重要

不能在任何 Linux 网络配置文件中配置出口 IP 地址，比如 ifcfg-eth0。

26.10.1.1. 平台支持

下表概述了对不同平台中的出口 IP 地址功能的支持：

平台	支持
裸机	是
VMware vSphere	是
Red Hat OpenStack Platform(RHOSP)	是
Amazon Web Services (AWS)	是
Google Cloud Platform (GCP)	是
Microsoft Azure	是
IBM Z® 和 IBM® LinuxONE	是
IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	是
IBM Power®	是
Nutanix	是

重要

在 Amazon Web Services(AWS)上置备的集群中不支持使用 EgressIP 功能将出口 IP 地址分配给 control plane 节点。(BZ#2039656).

26.10.1.2. 公共云平台注意事项

对于在公共云基础架构上置备的集群，每个节点绝对的 IP 地址会有一个约束。如下公式描述了每个节点的可分配 IP 地址或 *IP 容量*上限：

```
IP capacity = public cloud default capacity - sum(current IP assignments)
```

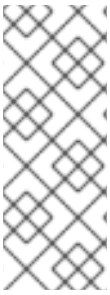
虽然 Egress IP 功能管理每个节点的 IP 地址容量，但在部署中计划这个约束非常重要。例如，对于在具有 8 个节点的裸机基础架构上安装的集群，您可以配置 150 个出口 IP 地址。但是，如果公共云提供商将 IP 地址容量限制为每个节点 10 个 IP 地址，则可分配 IP 地址总数仅为 80。为了在这个示例中获得相同的 IP 地址容量，您需要分配 7 个节点。

要确认公共云环境中任何节点的 IP 容量和子网，您可以输入 **oc get node <node_name> -o yaml** 命令。**cloud.network.openshift.io/egress-ipconfig** 注解包括节点的容量和子网信息。

注解值是一个带有单个对象的数组，其中包含为主网络接口提供以下信息的字段：

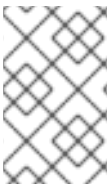
- **interface** ：指定 AWS 和 Azure 上的接口 ID，以及 GCP 上的接口名称。
- **ifaddr** ：为一个或多个 IP 地址系列指定子网掩码。
- **capacity** ：指定节点的 IP 地址容量。在 AWS 上，IP 地址容量为每个 IP 地址系列提供。在 Azure 和 GCP 上，IP 地址容量同时包括 IPv4 和 IPv6 地址。

为节点之间的流量自动附加和分离出口 IP 地址。这允许命名空间中许多 pod 的流量在集群外的位置上具有一致的源 IP 地址。这还支持 OpenShift SDN 和 OVN-Kubernetes，这是 OpenShift Container Platform 4.16 中 Red Hat OpenShift Networking 中的默认网络插件。



注意

RHOSP 出口 IP 地址功能会创建一个名为 **egressip-<IP address>** 的 neutron 保留端口。使用与 OpenShift Container Platform 集群安装相同的 RHOSP 用户，您可以为此保留端口分配一个浮动 IP 地址，以便为出口流量具有可预测的 SNAT 地址。当 RHOSP 网络上的出口 IP 地址从一个节点移到另一个节点时，因为节点故障转移（例如，neutron 保留端口会被删除并重新创建）。这意味着浮动 IP 关联丢失，您需要手动将浮动 IP 地址重新分配给新的保留端口。



注意

当 RHOSP 集群管理员为保留端口分配一个浮动 IP 时，OpenShift Container Platform 无法删除保留端口。**CloudPrivateIPConfig** 对象无法执行删除和移动操作，直到 RHOSP 集群管理员从保留端口取消分配浮动 IP。

以下示例演示了来自多个公共云提供商上节点的注解。注解被缩进以便于阅读。

AWS 上的 cloud.network.openshift.io/egress-ipconfig 注解示例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

GCP 上的 cloud.network.openshift.io/egress-ipconfig 注解示例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
```

```
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

以下小节描述了支持公共云环境的 IP 地址容量，用于容量计算。

26.10.1.2.1. Amazon Web Services(AWS)IP 地址容量限制

在 AWS 上，IP 地址分配的限制取决于配置的实例类型。如需更多信息，请参阅 每个实例类型的每个网络接口的 IP 地址

26.10.1.2.2. Google Cloud Platform(GCP)IP 地址容量限制

在 GCP 中，网络模型通过 IP 地址别名而不是 IP 地址分配来实施额外的节点 IP 地址。但是，IP 地址容量直接映射到 IP 别名容量。

IP 别名分配存在以下容量限制：

- 每个节点，IPv4 和 IPv6 的最大 IP 别名数为 100 个。
- 对于每个 VPC，IP 别名的最大数量没有被指定，但 OpenShift Container Platform 可扩展性测试显示最大为 15,000 个。

如需更多信息，请参阅 Per instance 配额和 Alias IP 范围概述。

26.10.1.2.3. Microsoft Azure IP 地址容量限制

在 Azure 上，IP 地址分配有以下容量限制：

- 对于每个 NIC，对于 IPv4 和 IPv6，可分配 IP 地址的最大数量为 256。
- 对于每个虚拟网络，分配的 IP 地址的最大数量不能超过 65,536。

如需更多信息，请参阅网络限制。

26.10.1.3. 在额外网络接口中使用出口 IP 的注意事项

在 OpenShift Container Platform 中，出口 IP 为管理员提供了一种控制网络流量的方法。出口 IP 可以与 br-ex 或主网络接口一起使用，它是与 Open vSwitch 关联的 Linux 网桥接口，或者可与额外的网络接口一起使用。

您可以运行以下命令来检查网络接口类型：

```
$ ip -details link show
```

主网络接口被分配一个节点 IP 地址，该地址还包含子网掩码。此节点 IP 地址的信息可以通过检查 k8s.ovn.org/node-primary-ifaddr 注解从集群中的每个节点检索。在 IPv4 集群中，此注解类似以下示例："k8s.ovn.org/node-primary-ifaddr: {"ipv4":"192.168.111.23/24"}"。

如果出口 IP 不在主网络接口子网的子网中，您可以在不是主网络接口类型的另一个 Linux 网络接口中使用出口 IP。这样一来，OpenShift Container Platform 管理员提供了对网络方面的更高级别控制，如路由、寻址、分段和安全策略。此功能允许用户选择通过特定网络接口路由工作负载流量，如流量分段或满足特殊要求。

如果出口 IP 不在主网络接口的子网中，如果节点上存在另一个网络接口，则可能会出现为出口流量选择另一个网络接口。

您可以通过检查 k8s.ovn.org/host-cidrs Kubernetes 节点注解来确定其他哪些网络接口可能支持出口 IP。此注释包含为主网络接口找到的地址和子网掩码。它还包含其他网络接口地址和子网掩码信息。这些地址和子网掩码分配给使用最长前缀匹配路由机制的网络接口，以确定哪个网络接口支持出口 IP。

注意

OVN-Kubernetes 提供了一种机制来控制和直接从特定命名空间和 pod 出站网络流量。这样可确保它通过特定的网络接口和特定的出口 IP 地址退出集群。

为不是主网络接口的网络接口分配出口 IP 的要求
对于希望出口 IP 和流量通过不是主网络接口的特定接口路由的用户，必须满足以下条件：

- OpenShift Container Platform 安装在裸机集群中。此功能在云或 hypervisor 环境中被禁用。
- 您的 OpenShift Container Platform pod 没有配置为 host-networked。
- 如果删除了网络接口，或者 IP 地址和子网掩码允许删除在接口上托管的出口 IP，则会重新配置出口 IP。因此，它可以分配给另一个节点和接口。
- 对于网络接口，必须启用 IP 转发。要启用 IP 转发，您可以使用 oc edit network.operator 命令并编辑类似以下示例的对象：

```
# ...
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
# ...
```

26.10.1.4. 将出口 IP 分配给 pod

要将一个或多个出口 IP 分配给命名空间中的命名空间或特定 pod,必须满足以下条件：

- 集群中至少有一个节点必须具有 k8s.ovn.org/egress-assignable: "" 标签。
- 存在一个 EgressIP 对象定义一个或多个出口 IP 地址，用作从命名空间中离开集群的流量的源 IP 地址。

重要

如果您在为出口 IP 分配标记集群中的任何节点之前创建 EgressIP 对象，OpenShift Container Platform 可能会将每个出口 IP 地址分配给第一个节点，并使用 k8s.ovn.org/egress-assignable: "" 标签。

要确保出口 IP 地址在集群中的不同节点广泛分发，请在创建任何 EgressIP 对象前，始终将标签应用到您想托管出口 IP 地址的节点。

26.10.1.5. 将出口 IP 分配给节点

在创建 EgressIP 对象时，以下条件适用于标记为 k8s.ovn.org/egress-assignable: "" 标签的节点：

- 每次不会将出口 IP 地址分配给多个节点。
- 出口 IP 地址可在可以托管出口 IP 地址的可用节点之间平衡。
- 如果 EgressIP 对象中的 spec.EgressIPs 数组指定了多个 IP 地址，则适用以下条件：
  - 任何节点都不会托管超过一个指定的 IP 地址。
  - 流量在给定命名空间的指定 IP 地址之间大致相等。
- 如果节点不可用，则会自动重新分配给它的所有出口 IP 地址，但符合前面描述的条件。

当 Pod 与多个 EgressIP 对象的选择器匹配时，无法保证在 EgressIP 对象中指定的出口 IP 地址被分配为 pod 的出口 IP 地址。

另外，如果 EgressIP 对象指定了多个出口 IP 地址，则无法保证可以使用哪些出口 IP 地址。例如，如果 pod 与带有两个出口 IP 地址 (10.10.20.1 和 10.10.20.2) 的 EgressIP 对象的选择器匹配，其中任何一个都可以用于每个 TCP 连接或 UDP 对话。

26.10.1.6. 出口 IP 地址配置架构图

下图显示了出口 IP 地址配置。图中描述了，在一个集群的三个节点上运行的两个不同命名空间中的四个 pod。节点从主机网络上的 192.168.126.0/18 CIDR 块中分配 IP 地址。

Node 1 和 Node 3 都标记为 k8s.ovn.org/egress-assignable: ""，因此可用于分配出口 IP 地址。

图中的横线描述了 pod1、pod2 和 pod 3 的流量流，通过 pod 网络来从 Node 1 和 Node 3 出口集群。当外部服务从示例 EgressIP 对象选择的任何 pod 接收流量时，源 IP 地址为 192.168.126.10 或 192.168.126.102。这两个节点之间流量大致平衡。

图中的以下资源被详细描述：

命名空间对象

命名空间在以下清单中定义：

命名空间对象

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1
kind: Namespace
metadata:
  name: namespace2
  labels:
    env: prod
```

EgressIP 对象

以下 EgressIP 对象描述了一个配置，该配置选择将 env 标签设置为 prod 的任意命名空间中的所有 pod。所选 pod 的出口 IP 地址为 192.168.126.10 和 192.168.126.102。

EgressIP 对象

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egressips-prod
spec:
  egressIPs:
  - 192.168.126.10
  - 192.168.126.102
  namespaceSelector:
    matchLabels:
      env: prod
status:
  items:
  - node: node1
    egressIP: 192.168.126.10
  - node: node3
    egressIP: 192.168.126.102
```

对于上例中的配置，OpenShift Container Platform 会为可用节点分配两个出口 IP 地址。status 字段显示是否以及在哪里分配了出口 IP 地址。

26.10.2. EgressIP 对象

以下 YAML 描述了 EgressIP 对象的 API。对象有效的范围为集群，它不是在命名空间中创建的。

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: <name> 1
spec:
  egressIPs: 2
  - <ip_address>
  namespaceSelector: 3
```

```
  ...
 podSelector: 4
  ...
```

1 EgressIPs 对象的名称。

2 包括一个或多个 IP 地址的数组。

3 出口 IP 地址与其关联的一个或多个命名空间选择器将。

4 可选：指定命名空间中的 pod 的一个或多个选择器，以将出口 IP 地址与其关联。通过使用这些选择器，可以选择命名空间中的 pod 子集。

以下 YAML 描述了命名空间选择器的小节：

命名空间选择器小节

```
namespaceSelector: 1
 matchLabels:
  <label_name>: <label_value>
```

1 命名空间的一个或多个匹配规则。如果提供多个匹配规则，则会选择所有匹配的命名空间。

以下 YAML 描述了 pod 选择器的可选小节：

Pod 选择器片段

```
podSelector: 1
 matchLabels:
  <label_name>: <label_value>
```

1 可选：与指定 namespaceSelector 规则匹配的命名空间中 pod 的一个或多个匹配规则。如果指定，则仅选择匹配的 pod。命名空间中的其他 Pod 不会被选择。

在以下示例中，EgressIP 对象将 192.168.126.11 和 192.168.126.102 出口 IP 地址与将 app 标签设置为 web 的 pod 关联，并位于将 env 标签设置为 prod 的命名空间中：

EgressIP 对象示例

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
 name: egress-group1
spec:
 egressIPs:
 - 192.168.126.11
 - 192.168.126.102
 podSelector:
  matchLabels:
   app: web
```

```
  namespaceSelector:
    matchLabels:
      env: prod
```

在以下示例中，EgressIP 对象将 192.168.127.30 和 192.168.127.40 出口 IP 地址与任何没有将 environment 标签设置为 development 的 pod 相关联：

EgressIP 对象示例

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group2
spec:
  egressIPs:
  - 192.168.127.30
  - 192.168.127.40
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: NotIn
      values:
      - development
```

26.10.3. EgressIPconfig 对象

作为出口 IP 的功能，reachabilityTotalTimeoutSeconds 参数配置探测发送到出口 IP 节点的检查的总超时时间。egressIPConfig 对象允许用户设置 reachabilityTotalTimeoutSeconds spec。如果在这个超时时间内无法访问 EgressIP 节点，则会声明该节点。

如果您的网络不够稳定以处理当前的默认值 1 秒，您可以提高这个值。

以下 YAML 描述了将 reachabilityTotalTimeoutSeconds 从默认的 1 秒探测改为 5 秒探测：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      egressIPConfig: 1
        reachabilityTotalTimeoutSeconds: 5 2
      gatewayConfig:
        routingViaHost: false
      genevePort: 6081
```

1. egressIPConfig 包含 EgressIP 对象选项的配置。通过更改这些配置，您可以扩展 EgressIP 对象。
2. reachabilityTotalTimeoutSeconds 的值接受从 0 到 60 的整数值。值 0 代表禁用 egressIP 节点的可达性检查。1 到 60 的值对应于发送节点的可达性检查之间的持续时间（以秒为单位）。

26.10.4. 标记节点以托管出口 IP 地址

您可以将 k8s.ovn.org/egress-assignable="" 标签应用到集群中的节点，以便 OpenShift Container Platform 可以为该节点分配一个或多个出口 IP 地址。

先决条件

- 安装 OpenShift CLI（oc）。
- 以集群管理员身份登录集群。

流程

- 要标记节点，使其可以托管一个或多个出口 IP 地址，请输入以下命令：

```
$ oc label nodes <node_name> k8s.ovn.org/egress-assignable="" 1
```

1 要标记的节点的名称。

提示

您还可以应用以下 YAML 将标签添加到节点：

```
apiVersion: v1
kind: Node
metadata:
  labels:
    k8s.ovn.org/egress-assignable: ""
  name: <node_name>
```

26.10.5. 后续步骤

- 分配出口 IP

26.10.6. 其他资源

- labelSelector meta/v1
- LabelSelectorRequirement meta/v1

26.11. 分配出口 IP 地址

作为集群管理员，您可以为从一个命名空间中，或从一个命名空间内的特定 pod 中离开集群的网络流量分配一个出口 IP 地址。

26.11.1. 为一个命名空间分配出口 IP 地址

您可以将一个或多个出口 IP 地址分配给一个命名空间，或分配给命名空间中的特定 pod。

先决条件

- 安装 OpenShift CLI（oc）。

- 以集群管理员身份登录集群。
- 至少配置一个节点来托管出口 IP 地址。

流程

1. 创建 EgressIP 对象：
   a. 创建一个 <egressips_name>.yaml 文件，其中 <egressips_name> 是对象的名称。
   b. 在您创建的文件中，定义一个 EgressIP 对象，如下例所示：

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-project1
spec:
  egressIPs:
  - 192.168.127.10
  - 192.168.127.11
  namespaceSelector:
    matchLabels:
      env: qa
```

2. 运行以下命令来创建对象。

```
$ oc apply -f <egressips_name>.yaml 1
```

1 将 <egressips_name> 替换为对象的名称。

输出示例

```
egressips.k8s.ovn.org/<egressips_name> created
```

3. 可选：保存 <egressips_name>.yaml 文件，以便在以后进行修改。
4. 为需要出口 IP 地址的命名空间添加标签。要在第 1 步中定义的 EgressIP 对象的命名空间中添加标签，请运行以下命令：

```
$ oc label ns <namespace> env=qa 1
```

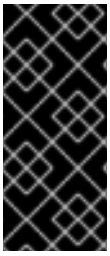
1 将 <namespace> 替换为需要出口 IP 地址的命名空间。

26.11.2. 其他资源

- 配置出口 IP 地址

26.12. 配置出口服务

作为集群管理员，您可以使用 egress 服务为负载均衡器服务后面的 pod 配置出口流量。



重要

出口服务只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以使用以下方法使用 EgressService 自定义资源(CR)管理出口流量：

- 将负载均衡器服务 IP 地址分配为负载均衡器服务后面的 pod 的源 IP 地址。
  将负载均衡器 IP 地址分配在这个上下文中的源 IP 地址对于显示单个出口和入口点很有用。例如，在某些情况下，与负载均衡器服务后面的应用程序通信的外部系统可能会预期应用程序的源和目标 IP 地址相同。

  注意

  当您将负载均衡器服务 IP 地址分配给服务后面的 pod 的出口流量时，OVN-Kubernetes 会将入口和出口点限制为单一节点。这限制了 MetalLB 通常提供的流量的负载均衡。

- 将负载均衡器后面的 pod 的出口流量分配给与默认节点网络不同的网络。
  这可用于将负载均衡器后面的应用程序的出口流量分配给与默认网络不同的网络。通常，不同的网络通过使用与网络接口关联的 VRF 实例来实施。

26.12.1. 出口服务自定义资源

在 EgressService 自定义资源中定义出口服务的配置。以下 YAML 描述了配置出口服务的字段：

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: <egress_service_name> 1
  namespace: <namespace> 2
spec:
  sourceIPBy: <egress_traffic_ip> 3
  nodeSelector: 4
    matchLabels:
      node-role.kubernetes.io/<role>: ""
  network: <egress_traffic_network> 5
```

1. 指定出口服务的名称。EgressService 资源的名称必须与您要修改的负载均衡器服务的名称匹配。
2. 指定出口服务的命名空间。EgressService 的命名空间必须与您要修改的负载均衡器服务的命名空间匹配。egress 服务是命名空间范围的。
3. 为服务后面的 pod 指定出口流量的源 IP 地址。有效值为 LoadBalancerIP 或 Network。使用 LoadBalancerIP 值将 LoadBalancer 服务入口 IP 地址分配为出口流量的源 IP 地址。指定 Network 将网络接口 IP 地址分配为出口流量的源 IP 地址。
4. 可选：如果您将 LoadBalancerIP 值用于 sourceIPBy 规格，则单一节点处理 LoadBalancer 服务流量。使用 nodeSelector 字段来限制哪些节点可以被分配。当选择节点来处理服务流量时，OVN-Kubernetes 以以下格式标记节点：egress-service.k8s.ovn.org/<svc-namespace>-<svc-name>: ""。如果没有指定 nodeSelector 字段，任何节点都可以管理 LoadBalancer 服务流量。

5 可选：指定出口流量的路由表。如果没有 network 规格，egress 服务将使用默认主机网络。

出口服务规格示例

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: test-egress-service
  namespace: test-namespace
spec:
  sourceIPBy: "LoadBalancerIP"
  nodeSelector:
    matchLabels:
      vrf: "true"
  network: "2"
```

26.12.2. 部署出口服务

您可以部署出口服务，以管理 LoadBalancer 服务后面的 pod 的出口流量。

以下示例将出口流量配置为具有与 LoadBalancer 服务的入口 IP 地址相同的源 IP 地址。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 已配置了 MetalLB BGPPeer 资源。

流程

1. 为服务创建一个带有所需 IP 的 IPAddressPool CR：
   a. 创建一个文件，如 ip-addr-pool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example-pool
  namespace: metallb-system
spec:
  addresses:
  - 172.19.0.100/32
```

   b. 运行以下命令，为 IP 地址池应用配置：

```
$ oc apply -f ip-addr-pool.yaml
```

2. 创建 Service 和 EgressService CR：
   a. 创建一个文件，如 service-egress-service.yaml，其内容类似以下示例：

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
  annotations:
    metallb.universe.tf/address-pool: example-pool 1
spec:
  selector:
    app: example
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
---
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: example-service
  namespace: example-namespace
spec:
  sourceIPBy: "LoadBalancerIP" 2
  nodeSelector: 3
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

1. LoadBalancer 服务使用 MetalLB 从example-pool IP 地址池分配的 IP 地址。
2. 本例使用 LoadBalancerIP 值来分配 LoadBalancer 服务的入口 IP 地址，作为出口流量的源 IP 地址。
3. 当您指定 LoadBalancerIP 值时，单个节点处理 LoadBalancer 服务的流量。在本例中，只能选择具有 worker 标签的节点来处理流量。当选择节点时，OVN-Kubernetes 会以 egress-service.k8s.ovn.org/<svc-namespace>-<svc-name>: "" 格式标记节点。

注意

如果使用 sourceIPBy: "LoadBalancerIP" 设置，您必须在 BGPAdvertisement 自定义资源(CR) 中指定 load-balancer 节点。

b. 运行以下命令，为服务和出口服务应用配置：

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

3. 创建 BGPAdvertisement CR 来公告服务：

a. 创建一个文件，如 service-bgp-advertisement.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
```

```
metadata:
  name: example-bgp-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - example-pool
  nodeSelector:
  - matchLabels:
      egress-service.k8s.ovn.org/example-namespace-example-service: "" (1)
```

(1) 在本例中，EgressService CR 将出口流量的源 IP 地址配置为使用负载均衡器服务 IP 地址。因此，您必须指定负载均衡器节点返回流量，以便为来自 pod 的流量使用相同的返回路径。

验证

1. 运行以下命令，验证您可以访问 MetalLB 服务后运行的 pod 的应用程序端点：

```
$ curl <external_ip_address>:<port_number> (1)
```

(1) 更新外部 IP 地址和端口号，以适合您的应用程序端点。

2. 如果您将 LoadBalancer 服务的入口 IP 地址分配为出口流量的源 IP 地址，请使用 tcpdump 等工具分析外部客户端接收的数据包来验证此配置。

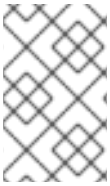
其他资源

- 通过网络 VRF 公开服务
- 示例：带有 VRF 实例网络配置策略的网络接口
- 使用 MetalLB 管理对称路由
- 关于虚拟路由和转发

26.13. 使用出口路由器 POD 的注意事项

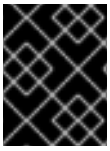
26.13.1. 关于出口路由器 pod

OpenShift Container Platform 出口路由器（egress router） pod 使用一个来自专用的私有源 IP 地址，将网络流量重定向到指定的远程服务器。出口路由器 pod 可以将网络流量发送到设置为仅允许从特定 IP 地址访问的服务器。



注意

出口路由器 pod 并不适用于所有外向的连接。创建大量出口路由器 pod 可能会超过您的网络硬件的限制。例如，为每个项目或应用程序创建出口路由器 pod 可能会导致，在转换为使用软件来进行 MAC 地址过滤前超过了网络接口可以处理的本地 MAC 地址的数量。



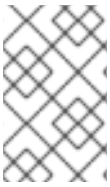
重要

出口路由器镜像与 Amazon AWS、Azure Cloud 或其他不支持第 2 层操作的云平台不兼容，因为它们与 macvlan 流量不兼容。

26.13.1.1. 出口路由器模式

在*重定向模式*中，出口路由器 Pod 配置 iptables 规则，将流量从其自身 IP 地址重定向到一个或多个目标 IP 地址。需要使用保留源 IP 地址的客户端 pod 必须配置为访问出口路由器的服务，而不是直接连接到目标 IP。您可以使用 curl 命令从应用程序 pod 访问目标服务和端口。例如：

```
$ curl <router_service_IP> <port>
```



注意

egress router CNI 插件只支持重定向模式。这与您可以使用 OpenShift SDN 部署的出口路由器实现不同。与 OpenShift SDN 的出口路由器不同，egress router CNI 插件不支持 HTTP 代理模式或 DNS 代理模式。

26.13.1.2. 出口路由器 pod 的实现

出口路由器实施使用出口路由器 Container Network Interface（CNI）插件。该插件将二级网络接口添加到 pod。

出口路由器是一个带有两个网络接口的 pod。例如，pod 可以具有 eth0 和 net1 网络接口。eth0 接口位于集群网络中，pod 将继续将接口用于与集群相关的普通网络流量。net1 接口位于第二个网络中，它拥有那个网络的 IP 地址和网关。OpenShift Container Platform 集群中的其他 pod 可以访问出口路由器服务，服务使 pod 可以访问外部服务。出口路由器作为 pod 和外部系统间的桥接。

离开出口路由器的流量会通过一个节点退出，但数据包带有来自路由器 pod 的 net1 接口的 MAC 地址。

添加出口路由器自定义资源时，Cluster Network Operator 会创建以下对象：

- pod 的 net1 二级网络接口的网络附加定义。
- 出口路由器的部署。

如果您删除了一个出口路由器自定义资源，Operator 会删除上列表中与出口路由器关联的两个对象。

26.13.1.3. 部署注意事项

出口路由器 pod 会为节点的主网络接口添加额外的 IP 地址和 MAC 地址。因此，您可能需要配置虚拟机监控程序或云供应商来允许额外的地址。

Red Hat OpenStack Platform (RHOSP)

如果在 RHOSP 上部署 OpenShift Container Platform，则必须允许来自 OpenStack 环境上的出口路由器 Pod 的 IP 和 MAC 地址的流量。如果您不允许流量，则通信会失败：

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

VMware vSphere

如果您使用 VMware vSphere，请参阅 VMware 文档来保护 vSphere 标准交换机。通过从 vSphere Web 客户端中选择主机虚拟交换机来查看并更改 VMware vSphere 默认设置。

具体来说，请确保启用了以下功能：

- MAC 地址更改
- Forged Transits
- Promiscuous Mode Operation

26.13.1.4. 故障切换配置

为了避免停机，Cluster Network Operator 会将出口路由器 pod 部署为部署资源。部署名称为 egress-router-cni-deployment。与部署对应的 pod 具有 app=egress-router-cni 标签。

要为部署创建新服务，请使用 oc expose deployment/egress-router-cni-deployment --port <port_number> 命令或创建类似以下示例的文件：

```
apiVersion: v1
kind: Service
metadata:
  name: app-egress
spec:
  ports:
  - name: tcp-8080
    protocol: TCP
    port: 8080
  - name: tcp-8443
    protocol: TCP
    port: 8443
  - name: udp-80
    protocol: UDP
    port: 80
  type: ClusterIP
  selector:
    app: egress-router-cni
```

26.13.2. 其他资源

- 在重定向模式中部署出口路由器

26.14. 以重定向模式部署出口路由器 POD

作为集群管理员，您可以部署出口路由器 Pod，将流量重新指向来自保留源 IP 地址的指定目标 IP 地址。

出口路由器实施使用出口路由器 Container Network Interface（CNI）插件。

26.14.1. 出口路由器自定义资源

在出口路由器自定义资源中定义出口路由器 pod 的配置。以下 YAML 描述了以重定向模式配置出口路由器的字段：

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: <egress_router_name>
```

```
  namespace: <namespace> 1
spec:
  addresses: [ 2
    {
      ip: "<egress_router>", 3
      gateway: "<egress_gateway>" 4
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [ 5
      {
        destinationIP: "<egress_destination>",
        port: <egress_router_port>,
        targetPort: <target_port>, 6
        protocol: <network_protocol> 7
      },
      ...
    ],
    fallbackIP: "<egress_destination>" 8
  }
```

1. 可选： namespace 字段指定要在其中创建出口路由器的命名空间。如果您没有在文件或命令行中指定值，则会使用 default 命名空间。
2. address 字段指定要在第二个网络接口上配置的 IP 地址。
3. ip 字段指定节点用于出口路由器 pod 的物理网络中保留源 IP 地址和子网掩码。使用 CIDR 表示法指定 IP 地址和网络掩码。
4. gateway 字段指定网络网关的 IP 地址。
5. 可选： redirectRules 字段指定出口目的地 IP 地址、出口路由器端口和协议的组合。到指定端口和协议中的出口路由器的传入连接路由到目标 IP 地址。
6. 可选： targetPort 字段指定目标 IP 地址上的网络端口。如果没有指定此字段，流量将路由到它到达的同一网络端口。
7. protocol 字段支持 TCP、UDP 或 SCTP。
8. 可选： fallbackIP 字段指定目标 IP 地址。如果没有指定任何重定向规则，出口路由器会将所有流量发送到这个回退 IP 地址。如果您指定了重定向规则，则出口路由器将任何与规则中定义的网络端口的连接发送到这个回退 IP 地址。如果没有指定此字段，出口路由器会拒绝与规则中没有定义的网络端口的连接。

出口路由器规格示例

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: egress-router-redirect
spec:
  networkInterface: {
    macvlan: {
```

```
    mode: "Bridge"
  }
}
addresses: [
  {
    ip: "192.168.12.99/24",
    gateway: "192.168.12.1"
  }
]
mode: Redirect
redirect: {
  redirectRules: [
    {
      destinationIP: "10.0.0.99",
      port: 80,
      protocol: UDP
    },
    {
      destinationIP: "203.0.113.26",
      port: 8080,
      targetPort: 80,
      protocol: TCP
    },
    {
      destinationIP: "203.0.113.27",
      port: 8443,
      targetPort: 443,
      protocol: TCP
    }
  ]
}
```

26.14.2. 以重定向模式部署出口路由器

您可以部署出口路由器，将其自身保留源 IP 地址的流量重定向到一个或多个目标 IP 地址。

添加出口路由器后，需要使用保留源 IP 地址的客户端 pod 必须修改为连接到出口路由器，而不是直接连接到目标 IP。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建出口路由器定义。
2. 为确保其他 pod 可以找到出口路由器 pod 的 IP 地址，请创建一个使用出口路由器的服务，如下例所示：

```
apiVersion: v1
kind: Service
metadata:
```

```
  name: egress-1
spec:
  ports:
  - name: web-app
    protocol: TCP
    port: 8080
  type: ClusterIP
  selector:
    app: egress-router-cni 1
```

1 指定出口路由器的标签。显示的值由 Cluster Network Operator 添加，且不可配置。

创建服务后，您的 Pod 可以连接到该服务。出口路由器 pod 将流量重定向到目标 IP 地址中对应的端口。连接来自保留的源 IP 地址。

验证

要验证 Cluster Network Operator 是否启动了出口路由器，请完成以下步骤：

1. 查看 Operator 为出口路由器创建的网络附加定义：

```
$ oc get network-attachment-definition egress-router-cni-nad
```

网络附加定义的名称不可配置。

输出示例

```
NAME                 AGE
egress-router-cni-nad   18m
```

2. 查看出口路由器 pod 的部署：

```
$ oc get deployment egress-router-cni-deployment
```

部署的名称不可配置。

输出示例

```
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
egress-router-cni-deployment   1/1     1            1           18m
```

3. 查看出口路由器 pod 的状态：

```
$ oc get pods -l app=egress-router-cni
```

输出示例

```
NAME                                         READY   STATUS    RESTARTS   AGE
egress-router-cni-deployment-575465c75c-qkq6m   1/1     Running   0          18m
```

4. 查看出口路由器 pod 的日志和路由表。

a. 获取出口路由器 pod 的节点名称：

```
$ POD_NODENAME=$(oc get pod -l app=egress-router-cni -o jsonpath="
{.items[0].spec.nodeName}")
```

b. 在目标节点上进入一个 debug 会话。此步骤被实例化为一个名为 <node_name>-debug 的 debug pod:

```
$ oc debug node/$POD_NODENAME
```

c. 将 /host 设为 debug shell 中的根目录。debug pod 在 pod 中的/host 中挂载主机 的 root 文件系统。将根目录改为 /host，您可以从主机的可执行路径中运行二进制文件：

```
# chroot /host
```

d. 在 chroot 环境控制台中显示出口路由器日志：

```
# cat /tmp/egress-router-log
```

输出示例

```
2021-04-26T12:27:20Z [debug] Called CNI ADD
2021-04-26T12:27:20Z [debug] Gateway: 192.168.12.1
2021-04-26T12:27:20Z [debug] IP Source Addresses: [192.168.12.99/24]
2021-04-26T12:27:20Z [debug] IP Destinations: [80 UDP 10.0.0.99/30 8080 TCP
203.0.113.26/30 80 8443 TCP 203.0.113.27/30 443]
2021-04-26T12:27:20Z [debug] Created macvlan interface
2021-04-26T12:27:20Z [debug] Renamed macvlan to "net1"
2021-04-26T12:27:20Z [debug] Adding route to gateway 192.168.12.1 on macvlan
interface
2021-04-26T12:27:20Z [debug] deleted default route {Ifindex: 3 Dst: <nil> Src: <nil>
Gw: 10.128.10.1 Flags: [] Table: 254}
2021-04-26T12:27:20Z [debug] Added new default route with gateway 192.168.12.1
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i
eth0 -p UDP --dport 80 -j DNAT --to-destination 10.0.0.99
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i
eth0 -p TCP --dport 8080 -j DNAT --to-destination 203.0.113.26:80
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i
eth0 -p TCP --dport 8443 -j DNAT --to-destination 203.0.113.27:443
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat -o net1 -j SNAT --to-
source 192.168.12.99
```

当您启动出口路由器时，通过创建 EgressRouter 对象来启动出口路由器时，日志文件位置和日志记录级别不可配置，如下所述。

e. 在 chroot 环境控制台中获取容器 ID：

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

输出示例

```
CONTAINER
bac9fae69ddb6
```

f. 确定容器的进程 ID。在本例中，容器 ID 是 bac9fae69ddb6：

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

输出示例

```
68857
```

g. 输入容器的网络命名空间：

```
# nsenter -n -t 68857
```

h. 显示路由表：

```
# ip route
```

在以下示例输出中，net1 网络接口是默认路由。集群网络的流量使用 eth0 网络接口。192.168.12.0/24 网络的流量使用 net1 网络接口，并来自保留源 IP 地址 192.168.12.99。pod 将所有其他流量路由到网关的 IP 地址 192.168.12.1。不显示服务网络的路由。

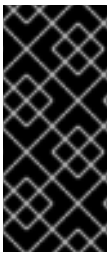
输出示例

```
default via 192.168.12.1 dev net1
10.128.10.0/23 dev eth0 proto kernel scope link src 10.128.10.18
192.168.12.0/24 dev net1 proto kernel scope link src 192.168.12.99
192.168.12.1 dev net1
```

26.15. 为项目启用多播

26.15.1. 关于多播

通过使用 IP 多播，数据可同时广播到许多 IP 地址。



重要

- 目前，多播最适用于低带宽协调或服务发现。它不是一个高带宽解决方案。
- 默认情况下，网络策略会影响命名空间中的所有连接。但是，多播不受网络策略的影响。如果在与网络策略相同的命名空间中启用了多播，则始终允许多播，即使有一个 deny-all 网络策略。在启用网络策略前，集群管理员应考虑对多播的影响。

默认情况下，OpenShift Container Platform pod 之间多播流量被禁用。如果使用 OVN-Kubernetes 网络插件，可以根据每个项目启用多播。

26.15.2. 启用 pod 间多播

您可以为项目启用 pod 间多播。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 运行以下命令，为项目启用多播。使用您要启用多播的项目的名称替换 <namespace>。

```
$ oc annotate namespace <namespace> \
    k8s.ovn.org/multicast-enabled=true
```

提示

您还可以应用以下 YAML 来添加注解：

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: "true"
```

验证

要验证项目是否启用了多播，请完成以下步骤：

1. 将您的当前项目更改为启用多播的项目。使用项目名替换 <project>。

```
$ oc project <project>
```

2. 创建 pod 以作为多播接收器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. 创建 pod 以作为多播发送器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
```

```
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 在新的终端窗口或选项卡中，启动多播监听程序。

   a. 获得 Pod 的 IP 地址：

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. 输入以下命令启动多播监听程序：

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. 启动多播传输。

   a. 获取 pod 网络 IP 地址范围：

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

   b. 要发送多播信息，请输入以下命令：

   ```
   $ oc exec msender -i -t -- \
       /bin/bash -c "echo | socat STDIO UDP4-
   DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
   ```

   如果多播正在工作，则上一个命令会返回以下输出：

   ```
   mlistener
   ```

26.16. 为项目禁用多播

26.16.1. 禁用 pod 间多播

您可以为项目禁用 pod 间多播。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 运行以下命令来禁用多播：

```
$ oc annotate namespace <namespace> \ 1
  k8s.ovn.org/multicast-enabled-
```

1 您要禁用多播的项目的 namespace。

提示

您还可以应用以下 YAML 来删除注解：

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: null
```

26.17. 跟踪网络流

作为集群管理员，您可以从集群中收集有关 pod 网络流的信息，以帮助以下区域：

- 监控 pod 网络上的入口和出口流量。
- 对性能问题进行故障排除。
- 为容量规划和安全审计收集数据。

当您启用网络流的集合时，只会收集与流量相关的元数据。例如，不会收集实际的数据包数据，而是只收集协议、源地址、目标地址、端口号、字节数和其他数据包级别的信息。

数据采用以下一种或多种记录格式收集：

- NetFlow
- sFlow
- IPFIX

当您使用一个或多个收集器 IP 地址和端口号配置 Cluster Network Operator（CNO）时，Operator 会在每个节点上配置 Open vSwitch（OVS），以将网络流记录发送到每个收集器。

您可以将 Operator 配置为将记录发送到多种类型的网络流收集器。例如，您可以将记录发送到 NetFlow 收集器，并将记录发送到 sFlow 收集器。

当 OVS 向收集器发送数据时，每种类型的收集器接收相同的记录。例如，如果您配置两个 NetFlow 收集器，节点上的 OVS 会将相同的记录发送到两个收集器。如果您还配置了两个 sFlow 收集器，则两个 sFlow 收集器将接收相同的记录。但是，每个收集器类型都具有唯一的记录格式。

收集网络流数据并将记录发送到收集器会影响性能。节点处理数据包的速度较慢。如果性能影响太大，您可以删除收集器的目的地，以禁用收集网络流数据并恢复性能。

注意

启用网络流收集器可能会影响集群网络的整体性能。

26.17.1. 用于跟踪网络流的网络对象配置

下表显示了在 Cluster Network Operator（CNO）中配置网络流收集器的字段：

表 26.18. 网络流配置

字段	类型	描述
metadata.name	**字符串**	CNO 对象的名称。这个名称始终是 **集群**。
spec.exportNetworkFlows	**object**	一个或多个 **netFlow**、**sFlow** 或 **ipfix**.
spec.exportNetworkFlows.netFlow.collectors	**数组**	最多 10 个收集器的 IP 地址和网络端口对列表。
spec.exportNetworkFlows.sFlow.collectors	**数组**	最多 10 个收集器的 IP 地址和网络端口对列表。
spec.exportNetworkFlows.ipfix.collectors	**数组**	最多 10 个收集器的 IP 地址和网络端口对列表。

将以下清单应用到 CNO 后，Operator 会在集群中的每个节点上配置 Open vSwitch（OVS），将网络流记录发送到侦听 **192.168.1.99:2056** 的 NetFlow 收集器。

跟踪网络流的配置示例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

26.17.2. 为网络流收集器添加目的地

作为集群管理器，您可以将 Cluster Network Operator（CNO）配置为发送有关 pod 网络的网络流元数据到网络流收集器。

先决条件

- 已安装 OpenShift CLI（**oc**）。

- 使用具有 cluster-admin 权限的用户登陆到集群。
- 您有一个网络流收集器，知道它所侦听的 IP 地址和端口。

流程

1. 创建补丁文件，用于指定网络流收集器类型以及收集器的 IP 地址和端口信息：

```
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

2. 使用网络流收集器配置 CNO：

```
$ oc patch network.operator cluster --type merge -p "$(cat <file_name>.yaml)"
```

输出示例

```
network.operator.openshift.io/cluster patched
```

验证

通常情况不需要进行验证。您可以运行以下命令，确认每个节点上的 Open vSwitch（OVS）已配置为将网络流记录发送到一个或多个收集器。

1. 查看 Operator 配置，确认配置了 exportNetworkFlows 字段：

```
$ oc get network.operator cluster -o jsonpath="{.spec.exportNetworkFlows}"
```

输出示例

```
{"netFlow":{"collectors":["192.168.1.99:2056"]}}
```

2. 查看每个节点中的 OVS 网络流配置：

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node -o
jsonpath='{range@.items[*]}{.metadata.name}{"\n"}{end}');
  do ;
    echo;
    echo $pod;
    oc -n openshift-ovn-kubernetes exec -c ovnkube-controller $pod \
      -- bash -c 'for type in ipfix sflow netflow ; do ovs-vsctl find $type ; done';
done
```

输出示例

```
ovnkube-node-xrn4p
_uuid               : a4d2aaca-5023-4f3d-9400-7275f92611f9
active_timeout      : 60
add_id_to_interface : false
engine_id           : []
```

```
engine_type        : []
external_ids       : {}
targets            : ["192.168.1.99:2056"]

ovnkube-node-z4vq9
_uuid              : 61d02fdb-9228-4993-8ff5-b27f01a29bd6
active_timeout     : 60
add_id_to_interface : false
engine_id          : []
engine_type        : []
external_ids       : {}
targets            : ["192.168.1.99:2056"]-

...
```

26.17.3. 删除网络流收集器的所有目的地

作为集群管理员，您可以配置 Cluster Network Operator（CNO）来停止将网络流元数据发送到网络流收集器。

先决条件

- 已安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。

流程

1. 删除所有网络流收集器：

```
$ oc patch network.operator cluster --type='json' \
    -p='[{"op":"remove", "path":"/spec/exportNetworkFlows"}]'
```

输出示例

```
network.operator.openshift.io/cluster patched
```

26.17.4. 其他资源

- Network [operator.openshift.io/v1]

26.18. 配置混合联网

作为集群管理员，您可以配置 Red Hat OpenShift Networking OVN-Kubernetes 网络插件，以允许 Linux 和 Windows 节点分别托管 Linux 和 Windows 工作负载。

26.18.1. 使用 OVN-Kubernetes 配置混合网络

您可以将集群配置为使用 OVN-Kubernetes 网络插件的混合网络。这允许支持不同节点网络配置的混合集群。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 权限的用户登陆到集群。
- 确保集群使用 OVN-Kubernetes 网络插件。

流程

1. 要配置 OVN-Kubernetes 混合网络覆盖，请输入以下命令：

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
  -p '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "hybridOverlayConfig":{
            "hybridClusterNetwork":[
              {
                "cidr": "<cidr>",
                "hostPrefix": <prefix>
              }
            ],
            "hybridOverlayVXLANPort": <overlay_port>
          }
        }
      }
    }
  }'
```

其中：

cidr

指定用于额外覆盖网络上节点的 CIDR 配置。这个 CIDR 无法与集群网络 CIDR 重叠。

hostPrefix

指定要分配给每个节点的子网前缀长度。例如，如果 hostPrefix 设为 23，则每个节点从 given cidr 中分配 a/23 子网，这样就能有 510（2^(32 - 23)- 2）个 pod IP 地址。如果需要从外部网络访问节点，请配置负载均衡器和路由器来管理流量。

hybridOverlayVXLANPort

为额外覆盖网络指定自定义 VXLAN 端口。这是在 vSphere 上安装的集群中运行 Windows 节点所需要的，且不得为任何其他云供应商配置。自定义端口可以是除默认 4789 端口外的任何打开的端口。有关此要求的更多信息，请参阅 Microsoft 文档中的 Pod 到主机间的 pod 连接性。

输出示例

```
network.operator.openshift.io/cluster patched
```

2. 要确认配置是活跃的，请输入以下命令。应用更新可能需要几分钟。

```
$ oc get network.operator.openshift.io -o jsonpath="
{.items[0].spec.defaultNetwork.ovnKubernetesConfig}"
```

26.18.2. 其他资源

- 使用自定义网络在 AWS 上安装集群
- 使用网络自定义在 Azure 上安装集群

第 27 章 OPENSHIFT SDN 网络插件

27.1. 关于 OPENSHIFT SDN 网络插件

OpenShift SDN 的一部分，OpenShift SDN 是一个网络插件，它使用软件定义型网络 (SDN) 方法来提供一个统一的集群网络，它允许 OpenShift Container Platform 集群间 pod 间的通信。此 pod 网络由 OpenShift SDN 建立和维护，它使用 Open vSwitch (OVS) 配置覆盖网络。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.1.1. OpenShift SDN 网络隔离模式

OpenShift SDN 提供三种 SDN 模式来配置 pod 网络：

- *网络策略*模式允许项目管理员使用 **NetworkPolicy** 对象配置自己的隔离策略。Network policy 是 OpenShift Container Platform 4.16 的默认模式。
- *多租户*模式为 Pod 和服务提供项目级别的隔离。来自不同项目的 Pod 不能与不同项目的 Pod 和服务互相发送或接收数据包。您可以针对项目禁用隔离，允许它将网络流量发送到整个集群中的所有 pod 和服务，并从那些 pod 和服务接收网络流量。
- *子网*模式提供一个扁平 pod 网络，每个 pod 都可以与所有其他 pod 和服务通信。网络策略模式提供与子网模式相同的功能。

27.1.2. 支持的网络插件功能列表

Red Hat OpenShift Networking 为网络插件(OpenShift SDN 和 OVN-Kubernetes)提供了两个选项，用于网络插件。下表总结了这两个网络插件的当前功能支持：

表 27.1. 默认 CNI 网络插件功能比较

功能	OpenShift SDN	OVN-Kubernetes
出口 IP	支持	支持
Egress 防火墙 [1]	支持	支持
出口路由器	支持	支持 [2]
混合网络	不支持	支持
IPsec 加密	不支持	支持
IPv6	不支持	支持 [3] [4]

功能	OpenShift SDN	OVN-Kubernetes
Kubernetes 网络策略	支持	支持
Kubernetes 网络策略日志	不支持	支持
多播	支持	支持
硬件卸载	不支持	支持

1. 在 OpenShift SDN 中，出口防火墙也称为出口网络策略。这和网络策略出口不同。
2. OVN-Kubernetes 的出口路由器仅支持重定向模式。
3. 只有裸机、vSphere、IBM Power®、IBM Z® 和 Red Hat OpenStack 集群才支持 IPv6。
4. IBM Power®、IBM Z® 和 Red Hat OpenStack 集群不支持 IPv6 单堆栈。

27.2. 为项目配置出口 IP

作为集群管理员，您可以配置 OpenShift SDN Container Network Interface (CNI) 网络插件，为项目分配一个或多个出口 IP 地址。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.2.1. 出口 IP 地址架构设计和实施

OpenShift Container Platform 出口 IP 地址功能可确保来自一个或多个命名空间中的一个或多个 pod 的流量具有集群网络之外的服务具有一致的源 IP 地址。

例如，您可能有一个 pod 定期查询托管在集群外服务器上的数据库。要强制对服务器进行访问要求，将数据包过滤设备配置为只允许来自特定 IP 地址的流量。为确保您可以可靠地允许从该特定 pod 访问服务器，您可以为向服务器发出请求的 pod 配置特定的出口 IP 地址。

分配给命名空间的出口 IP 地址与用来向特定目的地发送流量的出口路由器不同。

在一些集群配置中，应用程序 Pod 和入口路由器 pod 在同一个节点上运行。如果您在这种情况下为应用程序项目配置出口 IP 地址，当您向应用程序项目发送请求时，不会使用 IP 地址。

出口 IP 地址作为额外 IP 地址在节点的主网络接口中使用，且必须与节点的主 IP 地址位于同一个子网中。不能为集群中的任何其他节点分配额外的 IP 地址。

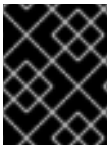
重要

不能在任何 Linux 网络配置文件中配置出口 IP 地址，比如 ifcfg-eth0。

27.2.1.1. 平台支持

下表概述了对不同平台中的出口 IP 地址功能的支持：

平台	支持
裸机	是
VMware vSphere	是
Red Hat OpenStack Platform(RHOSP)	是
Amazon Web Services (AWS)	是
Google Cloud Platform (GCP)	是
Microsoft Azure	是
IBM Z® 和 IBM® LinuxONE	是
IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	是
IBM Power®	是
Nutanix	是



重要

在 Amazon Web Services(AWS)上置备的集群中不支持使用 EgressIP 功能将出口 IP 地址分配给 control plane 节点。(BZ#2039656).

27.2.1.2. 公共云平台注意事项

对于在公共云基础架构上置备的集群，每个节点绝对的 IP 地址会有一个约束。如下公式描述了每个节点的可分配 IP 地址或 *IP 容量*上限：

```
IP capacity = public cloud default capacity - sum(current IP assignments)
```

虽然 Egress IP 功能管理每个节点的 IP 地址容量，但在部署中计划这个约束非常重要。例如，对于在具有 8 个节点的裸机基础架构上安装的集群，您可以配置 150 个出口 IP 地址。但是，如果公共云提供商将 IP 地址容量限制为每个节点 10 个 IP 地址，则可分配 IP 地址总数仅为 80。为了在这个示例中获得相同的 IP 地址容量，您需要分配 7 个节点。

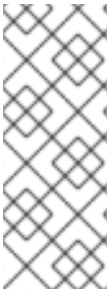
要确认公共云环境中任何节点的 IP 容量和子网，您可以输入 `oc get node <node_name> -o yaml` 命令。`cloud.network.openshift.io/egress-ipconfig` 注解包括节点的容量和子网信息。

注解值是一个带有单个对象的数组，其中包含为主网络接口提供以下信息的字段：

- `interface` ：指定 AWS 和 Azure 上的接口 ID，以及 GCP 上的接口名称。

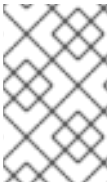
- **ifaddr** ：为一个或多个 IP 地址系列指定子网掩码。
- **capacity** ：指定节点的 IP 地址容量。在 AWS 上，IP 地址容量为每个 IP 地址系列提供。在 Azure 和 GCP 上，IP 地址容量同时包括 IPv4 和 IPv6 地址。

为节点之间的流量自动附加和分离出口 IP 地址。这允许命名空间中许多 pod 的流量在集群外的位置上具有一致的源 IP 地址。这还支持 OpenShift SDN 和 OVN-Kubernetes，这是 OpenShift Container Platform 4.16 中 Red Hat OpenShift Networking 中的默认网络插件。



注意

RHOSP 出口 IP 地址功能会创建一个名为 **egressip-<IP address>** 的 neutron 保留端口。使用与 OpenShift Container Platform 集群安装相同的 RHOSP 用户，您可以为此保留端口分配一个浮动 IP 地址，以便为出口流量具有可预测的 SNAT 地址。当 RHOSP 网络上的出口 IP 地址从一个节点移到另一个节点时，因为节点故障转移（例如，neutron 保留端口会被删除并重新创建）。这意味着浮动 IP 关联丢失，您需要手动将浮动 IP 地址重新分配给新的保留端口。



注意

当 RHOSP 集群管理员为保留端口分配一个浮动 IP 时，OpenShift Container Platform 无法删除保留端口。**CloudPrivateIPConfig** 对象无法执行删除和移动操作，直到 RHOSP 集群管理员从保留端口取消分配浮动 IP。

以下示例演示了来自多个公共云提供商上节点的注解。注解被缩进以便于阅读。

AWS 上的 cloud.network.openshift.io/egress-ipconfig 注解示例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

GCP 上的 cloud.network.openshift.io/egress-ipconfig 注解示例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

以下小节描述了支持公共云环境的 IP 地址容量，用于容量计算。

27.2.1.2.1. Amazon Web Services(AWS)IP 地址容量限制

在 AWS 上，IP 地址分配的限制取决于配置的实例类型。如需更多信息，请参阅 每个实例类型的每个网络接口的 IP 地址

27.2.1.2.2. Google Cloud Platform(GCP)IP 地址容量限制

在 GCP 中，网络模型通过 IP 地址别名而不是 IP 地址分配来实施额外的节点 IP 地址。但是，IP 地址容量直接映射到 IP 别名容量。

IP 别名分配存在以下容量限制：

- 每个节点，IPv4 和 IPv6 的最大 IP 别名数为 100 个。
- 对于每个 VPC，IP 别名的最大数量没有被指定，但 OpenShift Container Platform 可扩展性测试显示最大为 15,000 个。

如需更多信息，请参阅 Per instance 配额和 Alias IP 范围概述。

27.2.1.2.3. Microsoft Azure IP 地址容量限制

在 Azure 上，IP 地址分配有以下容量限制：

- 对于每个 NIC，对于 IPv4 和 IPv6，可分配 IP 地址的最大数量为 256。
- 对于每个虚拟网络，分配的 IP 地址的最大数量不能超过 65,536。

如需更多信息，请参阅网络限制。

27.2.1.3. 在额外网络接口中使用出口 IP 的注意事项

在 OpenShift Container Platform 中，出口 IP 为管理员提供了一种控制网络流量的方法。出口 IP 可以与 br-ex 或主网络接口一起使用，它是与 Open vSwitch 关联的 Linux 网桥接口，或者可与额外的网络接口一起使用。

您可以运行以下命令来检查网络接口类型：

```
$ ip -details link show
```

主网络接口被分配一个节点 IP 地址，该地址还包含子网掩码。此节点 IP 地址的信息可以通过检查 k8s.ovn.org/node-primary-ifaddr 注解从集群中的每个节点检索。在 IPv4 集群中，此注解类似以下示例："k8s.ovn.org/node-primary-ifaddr: {"ipv4":"192.168.111.23/24"}"。

如果出口 IP 不在主网络接口子网的子网中，您可以在不是主网络接口类型的另一个 Linux 网络接口中使用出口 IP。这样一来，OpenShift Container Platform 管理员提供了对网络方面的更高级别控制，如路由、寻址、分段和安全策略。此功能允许用户选择通过特定网络接口路由工作负载流量，如流量分段或满足特殊要求。

如果出口 IP 不在主网络接口的子网中，如果节点上存在另一个网络接口，则可能会出现为出口流量选择另一个网络接口。

您可以通过检查 k8s.ovn.org/host-cidrs Kubernetes 节点注解来确定其他哪些网络接口可能支持出口 IP。此注释包含为主网络接口找到的地址和子网掩码。它还包含其他网络接口地址和子网掩码信息。这些地址和子网掩码分配给使用最长前缀匹配路由机制的网络接口，以确定哪个网络接口支持出口 IP。

注意

OVN-Kubernetes 提供了一种机制来控制和直接从特定命名空间和 pod 出站网络流量。这样可确保它通过特定的网络接口和特定的出口 IP 地址退出集群。

为不是主网络接口的网络接口分配出口 IP 的要求
对于希望出口 IP 和流量通过不是主网络接口的特定接口路由的用户，必须满足以下条件：

- OpenShift Container Platform 安装在裸机集群中。此功能在云或 hypervisor 环境中被禁用。
- 您的 OpenShift Container Platform pod 没有配置为 host-networked。
- 如果删除了网络接口，或者 IP 地址和子网掩码允许删除在接口上托管的出口 IP，则会重新配置出口 IP。因此，它可以分配给另一个节点和接口。
- 对于网络接口，必须启用 IP 转发。要启用 IP 转发，您可以使用 oc edit network.operator 命令并编辑类似以下示例的对象：

```
# ...
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
# ...
```

== 限制

在 OpenShift SDN 网络插件中使用出口 IP 地址时会有以下限制：

- 您不能在同一节点上同时使用手动分配和自动分配的出口 IP 地址。
- 如果手动从 IP 地址范围分配出口 IP 地址，则不得将该范围用于自动 IP 分配。
- 您不能使用 OpenShift SDN 出口 IP 地址在多个命名空间间共享出口 IP 地址。

如果您需要在命名空间间共享 IP 地址，则 OVN-Kubernetes 网络插件出口 IP 地址可以跨越多个命名空间中的 IP 地址。

注意

如果您以多租户模式使用 OpenShift SDN，则无法将出口 IP 地址与与其关联的项目附加到另一个命名空间的任何命名空间一起使用。例如，如果 project1 和 project2 通过运行 oc adm pod-network join-projects --to=project1 project2 命令被连接，则这两个项目都不能使用出口 IP 地址。如需更多信息，请参阅 BZ#1645577。

27.2.1.4. IP 地址分配方法

您可以通过设置 NetNamespace 对象的 egressIPs 参数,将出口 IP 地址分配给命名空间。在出口 IP 地址与项目关联后，OpenShift SDN 允许您以两种方式为主机分配出口 IP 地址：

- 在*自动分配*方法中，给节点分配一个出口 IP 地址范围。
- 在*手动分配*方法中，给节点分配包含一个或多个出口 IP 地址的列表。

请求出口 IP 地址的命名空间与可以托管那些出口 IP 地址的节点匹配，然后为那些节点分配出口 IP 地址。如果在 NetNamespace 对象中设置了 egressIPs 参数，但没有节点托管该出口 IP 地址，则会丢弃来自该命名空间的出口流量。

节点高可用性是自动的。如果托管出口 IP 地址的节点不可访问，并且有可以托管那些出口 IP 地址的节点，那么出口 IP 地址将会移到新节点。当无法访问的托管原始出口 IP 地址的节点恢复正常后，出口 IP 地址会自动转移，以在不同节点之间均衡出口 IP 地址。

27.2.1.4.1. 使用自动分配的出口 IP 地址时的注意事项

当对出口 IP 地址使用自动分配方法时，请注意以下事项：

- 您可以设置每个节点的 HostSubnet 资源的 egressCIDRs 参数，以指明节点可以托管的出口 IP 地址范围。OpenShift Container Platform 根据您指定的 IP 地址范围设置 HostSubnet 资源的 egressIPs 参数。

如果托管命名空间的出口 IP 地址的节点不可访问，OpenShift Container Platform 会将出口 IP 地址重新分配给具有兼容出口 IP 地址范围的另外一个节点。自动分配方法最适合在把额外的 IP 地址与节点进行关联时具有灵活性的环境中安装的集群。

27.2.1.4.2. 使用手动分配出口 IP 地址时的注意事项

这种方法允许您控制哪些节点可以托管出口 IP 地址。

注意

如果在公共云基础架构上安装了集群，则必须确保为每个节点分配出口 IP 地址，以便有足够的备用容量来托管 IP 地址。如需更多信息，请参阅上一节中的"平台注意事项"。

当手动分配出口 IP 地址时，请考虑以下事项：

- 您可以设置每个节点的 HostSubnet 资源的 egressIPs 参数，以指明节点可以托管的 IP 地址。
- 支持一个命名空间带有多个出口 IP 地址。

如果命名空间有多个出口 IP 地址，且这些地址托管在多个节点上，则需要考虑以下额外的注意事项：

- 如果 pod 位于托管出口 IP 地址的节点上，则该 pod 始终使用该节点上的出口 IP 地址。
- 如果 pod 不在托管出口 IP 地址的节点上，则该 pod 会随机使用出口 IP 地址。

27.2.2. 为一个命名空间启用自动分配出口 IP 地址

在 OpenShift Container Platform 中，可以为一个或多个节点上的特定命名空间启用自动分配出口 IP 地址。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI(oc)。

流程

1. 使用以下 JSON，用出口 IP 地址更新 NetNamespace 资源：

```
$ oc patch netnamespace <project_name> --type=merge -p \
 '{
   "egressIPs": [
```

```
      "<ip_address>"
    ]
  }'
```

其中：

<project_name>

指定项目的名称。

<ip_address>

为 egressIPs 数组指定一个或多个出口 IP 地址。

例如，将 project1 分配给 IP 地址 192.168.1.100，将project2 分配给 IP 地址 192.168.1.101：

```
$ oc patch netnamespace project1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100"]}'
$ oc patch netnamespace project2 --type=merge -p \
  '{"egressIPs": ["192.168.1.101"]}'
```

注意

由于 OpenShift SDN 管理 NetNamespace 对象，因此只能通过修改现有的 NetNamespace 对象来进行更改。不要创建新的 NetNamespace 对象。

2. 使用以下 JSON 设置每一主机的 egressCIDRs 参数，以指明哪些节点可以托管出口 IP 地址：

```
$ oc patch hostsubnet <node_name> --type=merge -p \
  '{
    "egressCIDRs": [
      "<ip_address_range>", "<ip_address_range>"
    ]
  }'
```

其中：

<node_name>

指定节点名称。

<ip_address_range>

指定 CIDR 格式的 IP 地址范围。您可以为 egressCIDRs 阵列指定多个地址范围。

例如，将 node1 和 node2 设置为托管范围为 192.168.1.0 到 192.168.1.255 的出口 IP 地址：

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
```

OpenShift Container Platform 会自动以均衡的方式将特定的出口 IP 地址分配给可用的节点。在本例中，它会将出口 IP 地址 192.168.1.100 分配给 node1，并将出口 IP 地址 192.168.1.101 分配给 node2，或反之。

27.2.3. 为一个命名空间配置手动分配出口 IP 地址

在 OpenShift Container Platform 中，您可以将一个或多个出口 IP 与一个项目关联。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI(oc)。

流程

1. 通过使用所需 IP 地址指定以下 JSON 对象来更新 NetNamespace 对象：

```
$ oc patch netnamespace <project_name> --type=merge -p \
 '{
   "egressIPs": [
     "<ip_address>"
   ]
 }'
```

其中：

<project_name>
指定项目的名称。

<ip_address>
为 egressIPs 数组指定一个或多个出口 IP 地址。

例如，将 project1 项目分配给 IP 地址 192.168.1.100 和 192.168.1.101：

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100","192.168.1.101"]}'
```

要提供高可用性，将 egressIPs 值设置为不同节点上的两个或多个 IP 地址。如果设置了多个出口 IP 地址，则 pod 会大致同样使用所有出口 IP 地址。

注意

由于 OpenShift SDN 管理 NetNamespace 对象，因此只能通过修改现有的 NetNamespace 对象来进行更改。不要创建新的 NetNamespace 对象。

2. 手动将出口 IP 地址分配给节点主机。
如果在公共云基础架构上安装了集群，则必须确认该节点具有可用的 IP 地址容量。

在节点主机上的 HostSubnet 对象中设置 egressIPs 参数。使用以下 JSON，尽可能包含您要分配给该节点主机的 IP 地址：

```
$ oc patch hostsubnet <node_name> --type=merge -p \
 '{
   "egressIPs": [
     "<ip_address>",
     "<ip_address>"
    ]
 }'
```

其中：

<node_name>
指定节点名称。

<ip_address>
指定一个 IP 地址。您可以为 egressIPs 数组指定多个 IP 地址。

例如，指定 node1 应具有出口 IP 192.168.1.100、192.168.1.101 和 192.168.1.102：

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

在上例中，project1 的所有出口流量都将路由到托管指定出口 IP 地址的节点，然后通过网络地址转换（NAT）连接到那个 IP 地址。

27.2.4. 其他资源

- 如果要配置手动出口 IP 地址分配，请参阅平台考虑 与 IP 容量规划相关的信息。

27.3. 为项目配置出口防火墙

作为集群管理员，您可以为项目创建一个出口防火墙，用于限制离开 OpenShift Container Platform 集群的出口流量。

注意

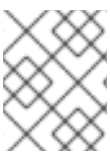
从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.3.1. 出口防火墙在一个项目中的工作原理

作为集群管理员，您可以使用一个*出口防火墙*来限制集群内的一些 pod 或所有 pod 可以访问的外部主机。出口防火墙适用于以下情况：

- pod 只能连接到内部主机，且无法启动到公共互联网的连接。
- pod 只能连接到公共互联网，且无法启动到 OpenShift Container Platform 集群以外的内部主机的连接。
- pod 无法访问 OpenShift Container Platform 集群外的特定内部子网或主机。
- pod 只能连接到特定的外部主机。

例如，您可以允许某一个项目访问指定的 IP 范围，但拒绝其他项目对同一 IP 范围的访问。或者您可以限制应用程序开发人员从 Python pip 的镜像点进行更新，并强制要求更新只能来自于批准的源。



注意

出口防火墙不适用于主机网络命名空间。启用主机网络的 Pod 不受出口防火墙规则的影响。

您可以通过创建一个 EgressNetworkPolicy 自定义资源（CR）对象来配置出口防火墙策略。出口防火墙与满足以下任一条件的网络流量匹配：

- CIDR 格式的 IP 地址范围
- 解析为 IP 地址的 DNS 名称

重要

如果您的出口防火墙包含 0.0.0.0/0 的拒绝规则，则阻止访问 OpenShift Container Platform API 服务器。您必须为每个 IP 地址添加允许规则，或使用出口策略规则中的 nodeSelector 类型允许规则来连接到 API 服务器。

以下示例演示了确保 API 服务器访问所需的出口防火墙规则的顺序：

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
  namespace: <namespace> 1
spec:
  egress:
  - to:
      cidrSelector: <api_server_address_range> 2
    type: Allow
# ...
  - to:
      cidrSelector: 0.0.0.0/0 3
    type: Deny
```

1. 出口防火墙的命名空间。
2. 包含 OpenShift Container Platform API 服务器的 IP 地址范围。
3. 一个全局拒绝规则会阻止访问 OpenShift Container Platform API 服务器。

要查找 API 服务器的 IP 地址，请运行 oc get ep kubernetes -n default。

如需更多信息，请参阅 BZ#1988324。

重要

您必须将 OpenShift SDN 配置为使用网络策略或多租户模式来配置出口防火墙。

如果您使用网络策略模式，则出口防火墙只与每个命名空间的一个策略兼容，且无法用于共享网络的项目，如全局项目。

警告

出口防火墙规则不适用于通过路由器的网络流量。任何有权创建 Route CR 对象的用户，都可以通过创建指向禁止的目的地的路由来绕过出口防火墙策略规则。

27.3.1.1. 出口防火墙的限制

出口防火墙有以下限制：

- 项目不能有多个 EgressNetworkPolicy 对象。

重要

允许创建多个 EgressNetworkPolicy 对象，但不应这样做。当您创建多个 EgressNetworkPolicy 对象时，会返回以下信息：**dropping all rules**。实际上，所有外部流量都会被丢弃，这可能会给您的组织造成安全风险。

- 每个项目最多可定义一个最多具有 1000 个规则的 EgressNetworkPolicy 对象。
- **default** 项目无法使用出口防火墙。
- 当在多租户模式下使用 OpenShift SDN 网络插件时，会有以下限制：
  - 全局项目无法使用出口防火墙。您可以使用 **oc adm pod-network make-projects-global** 把一个项目设置为全局项目。
  - 通过 **oc adm pod-network join-projects** 命令合并的项目，无法在任何合并的项目中使用出口防火墙。

违反这些限制会导致项目的出口防火墙出现问题。因此，所有外部网络流量都会被丢弃，这可能会给您的组织造成安全风险。

可在 **kube-node-lease**、**kube-public**、**kube-system**、**openshift** 和 **openshift-** 项目中创建一个 Egress Firewall 资源。

27.3.1.2. 出口防火墙策略规则的匹配顺序

出口防火墙策略规则按照它们定义的顺序来评估，从第一个到最后一个的顺序。第一个与 pod 的出口连接匹配的规则会被应用。该连接会忽略后续的所有规则。

27.3.1.3. 域名服务器 (DNS) 解析如何工作

如果您在 egress 防火墙策略规则中使用 DNS 名称，则正确解析域名会受到以下限制：

- 域名更新会根据生存时间（TTL）持续时间进行轮询。默认情况下，持续时间为 30 秒。当出口防火墙控制器查询本地名称服务器以获取域名时，如果响应中包含的 TTL 小于 30 秒，控制器会将持续时间设置为返回的值。如果响应中的 TTL 大于 30 分钟，控制器会将持续时间设置为 30 分钟。如果 TTL 介于 30 秒到 30 分钟之间，控制器会忽略该值，并将持续时间设置为 30 秒。
- 在需要时，pod 必须通过相同的本地名称服务器解析域名。否则，egress 防火墙控制器和 pod 已知的域的 IP 地址可能会有所不同。如果主机名的 IP 地址不同，则出口防火墙的强制实施可能不一致。
- 因为出口防火墙控制器和 pod 异步轮询相同的本地名称服务器，所以 pod 可能会在出口控制器执行前获取更新的 IP 地址，从而导致竞争条件。由于这个限制，仅建议在 EgressNetworkPolicy 对象中使用域名来更改 IP 地址的域。

注意

在出口防火墙策略中使用 DNS 名称不会影响通过 CoreDNS 的本地 DNS 解析。

但是，如果您的出口防火墙策略使用域名，并且外部 DNS 服务器处理受影响 pod 的 DNS 解析，则必须包括允许访问 DNS 服务器的 IP 地址的出口防火墙规则。

27.3.2. EgressNetworkPolicy 自定义资源 (CR) 对象

您可以为出口防火墙定义一个或多个规则。规则是一个 Allow 规则，也可以是一个 Deny 规则，它包括规则适用的流量规格。

以下 YAML 描述了一个 EgressNetworkPolicy CR 对象：

EgressNetworkPolicy 对象

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1 出口防火墙的名称。

2 以下部分所述，一个或多个出口网络策略规则的集合。

27.3.2.1. EgressNetworkPolicy 规则

以下 YAML 描述了一个出口防火墙规则对象。用户可以选择 CIDR 格式的 IP 地址范围、域名，或使用 nodeSelector 允许或拒绝出口流量。egress 小节需要一个包括一个或多个对象的数组。

出口策略规则小节

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns_name> 4
```

1 规则类型。该值必须是 Allow 或 Deny。

2 描述出口流量匹配规则的小节。规则的 cidrSelector 字段或 dnsName 字段的值。您不能在同一规则中使用这两个字段。

3 CIDR 格式的 IP 地址范围。

4 一个域名。

27.3.2.2. EgressNetworkPolicy CR 对象示例

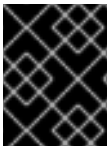
以下示例定义了几个出口防火墙策略规则：

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 出口防火墙策略规则对象的集合。

27.3.3. 创建出口防火墙策略对象

作为集群管理员，您可以为项目创建一个出口防火墙策略对象。



重要

如果项目已经定义了一个 EgressNetworkPolicy 对象，您必须编辑现有的策略来更改出口防火墙规则。

先决条件

- 使用 OpenShift SDN 网络插件的集群。
- 安装 OpenShift CLI (oc) 。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 创建策略规则：
   a. 创建一个 <policy_name>.yaml 文件，其中 <policy_name> 描述出口策略规则。
   b. 在您创建的文件中，定义出口策略对象。
2. 运行以下命令来创建策略对象。将 <policy_name> 替换为策略的名称， <project> 替换为规则应用到的项目。

```
$ oc create -f <policy_name>.yaml -n <project>
```

在以下示例中，在名为 project1 的项目中创建一个新的 EgressNetworkPolicy 对象：

```
$ oc create -f default.yaml -n project1
```

输出示例

```
egressnetworkpolicy.network.openshift.io/v1 created
```

3. 可选：保存 <policy_name>.yaml 文件，以便在以后进行修改。

27.4. 为项目编辑出口防火墙

作为集群管理员，您可以修改现有出口防火墙的网络流量规则。

27.4.1. 查看 EgressNetworkPolicy 对象

您可以查看集群中的 EgressNetworkPolicy 对象。

先决条件

- 使用 OpenShift SDN 网络插件的集群。
- 安装 OpenShift 命令行界面 (CLI)，通常称为 oc。
- 您必须登录集群。

流程

1. 可选：要查看集群中定义的 EgressNetworkPolicy 对象的名称，请输入以下命令：

```
$ oc get egressnetworkpolicy --all-namespaces
```

2. 要检查策略，请输入以下命令。将 <policy_name> 替换为要检查的策略名称。

```
$ oc describe egressnetworkpolicy <policy_name>
```

输出示例

```
Name:  default
Namespace: project1
Created: 20 minutes ago
Labels:  <none>
Annotations: <none>
Rule:  Allow to 1.2.3.0/24
Rule:  Allow to www.example.com
Rule:  Deny to 0.0.0.0/0
```

27.5. 为项目编辑出口防火墙

作为集群管理员，您可以修改现有出口防火墙的网络流量规则。

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.5.1. 编辑 EgressNetworkPolicy 对象

作为集群管理员，您可以更新一个项目的出口防火墙。

先决条件

- 使用 OpenShift SDN 网络插件的集群。
- 安装 OpenShift CLI (**oc**) 。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 查找项目的 EgressNetworkPolicy 对象的名称。将 **<project>** 替换为项目的名称。

```
$ oc get -n <project> egressnetworkpolicy
```

2. 可选，如果您在创建出口网络防火墙时没有保存 EgressNetworkPolicy 对象的副本，请输入以下命令来创建副本。

```
$ oc get -n <project> egressnetworkpolicy <name> -o yaml > <filename>.yaml
```

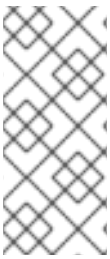
将 **<project>** 替换为项目的名称。将 **<name>** 替换为 Pod 的名称。将 **<filename>** 替换为要将 YAML 保存到的文件的名称。

3. 更改了策略规则后，请输入以下命令替换 EgressNetworkPolicy 对象。将 **<filename>** 替换为包含更新的 EgressNetworkPolicy 对象的文件名称。

```
$ oc replace -f <filename>.yaml
```

27.6. 从项目中删除出口防火墙

作为集群管理员，您可以从项目中删除出口防火墙，从而删除对项目的离开 OpenShift Container Platform 集群的网络流量的限制。



注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.6.1. 删除 EgressNetworkPolicy 对象

作为集群管理员，您可以从项目中删除出口防火墙。

先决条件

- 使用 OpenShift SDN 网络插件的集群。
- 安装 OpenShift CLI (oc) 。
- 您需要使用集群管理员身份登陆到集群。

流程

1. 查找项目的 EgressNetworkPolicy 对象的名称。将 <project> 替换为项目的名称。

```
$ oc get -n <project> egressnetworkpolicy
```

2. 输入以下命令删除 EgressNetworkPolicy 对象。将 <project> 替换为项目名称，<name> 替换为对象名称。

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

27.7. 使用出口路由器 POD 的注意事项

27.7.1. 关于出口路由器 pod

OpenShift Container Platform 出口路由器（egress router） pod 使用一个来自专用的私有源 IP 地址，将网络流量重定向到指定的远程服务器。出口路由器 pod 可以将网络流量发送到设置为仅允许从特定 IP 地址访问的服务器。

注意

出口路由器 pod 并不适用于所有外向的连接。创建大量出口路由器 pod 可能会超过您的网络硬件的限制。例如，为每个项目或应用程序创建出口路由器 pod 可能会导致，在转换为使用软件来进行 MAC 地址过滤前超过了网络接口可以处理的本地 MAC 地址的数量。

重要

出口路由器镜像与 Amazon AWS、Azure Cloud 或其他不支持第 2 层操作的云平台不兼容，因为它们与 macvlan 流量不兼容。

27.7.1.1. 出口路由器模式

在*重定向模式*中，出口路由器 Pod 配置 iptables 规则，将流量从其自身 IP 地址重定向到一个或多个目标 IP 地址。需要使用保留源 IP 地址的客户端 pod 必须配置为访问出口路由器的服务，而不是直接连接到目标 IP。您可以使用 curl 命令从应用程序 pod 访问目标服务和端口。例如：

```
$ curl <router_service_IP> <port>
```

在 *HTTP 代理模式*中，出口路由器 pod 作为一个 HTTP 代理在端口8080 上运行。这个模式只适用于连接到基于 HTTP 或基于 HTTPS 服务的客户端，但通常需要较少的更改就可以使客户端 pod 正常工作。很多程序可以通过设置环境变量来使用 HTTP 代理服务器。

在 *DNS 代理模式*中，出口路由器 pod 作为基于 TCP 服务的 DNS 代理运行，将其自身的 IP 地址转换到一个或多个目标 IP 地址。要使用保留的源 IP 地址，客户端 pod 必须进行修改来连接到出口路由器 pod，而不是直接连接到目标 IP 地址。此修改确保了外部的目的地将流量视为来自一个已知源的流量。

重定向模式可用于除 HTTP 和 HTTPS 以外的所有服务。对于 HTTP 和 HTTPS 服务，请使用 HTTP 代理模式。对于使用 IP 地址或域名的基于 TCP 的服务，请使用 DNS 代理模式。

27.7.1.2. 出口路由器 pod 的实现

出口路由器 pod 的设置由一个初始化容器执行。该容器在特权环境中运行，以便可以配置 macvlan 接口并设置 iptables 规则。在初始化容器完成设置 iptables 规则后会退出。接下来，出口路由器 pod 会执行容器来处理出口路由器流量。取决于出口路由器的模式，所使用的镜像会有所不同。

环境变量决定 egress-router 镜像使用的地址。镜像将 macvlan 接口配置为使用 EGRESS_SOURCE 作为其 IP 地址，并将 EGRESS_GATEWAY 作为网关的 IP 地址。

网络地址转换（NAT）规则被设置，使任何到 TCP 或 UDP 端口上的 pod 的集群 IP 地址的连接被重新指向由 EGRESS_DESTINATION 变量指定的 IP 地址的同一端口。

如果集群中只有部分节点能够声明指定的源 IP 地址并使用指定的网关，您可以指定一个 nodeName 或 nodeSelector 来表示哪些节点可以接受。

27.7.1.3. 部署注意事项

出口路由器 pod 会为节点的主网络接口添加额外的 IP 地址和 MAC 地址。因此，您可能需要配置虚拟机监控程序或云供应商来允许额外的地址。

Red Hat OpenStack Platform (RHOSP)

如果在 RHOSP 上部署 OpenShift Container Platform，则必须允许来自 OpenStack 环境上的出口路由器 Pod 的 IP 和 MAC 地址的流量。如果您不允许流量，则通信会失败：

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

VMware vSphere

如果您使用 VMware vSphere，请参阅 VMware 文档来保护 vSphere 标准交换机。通过从 vSphere Web 客户端中选择主机虚拟交换机来查看并更改 VMware vSphere 默认设置。

具体来说，请确保启用了以下功能：

- MAC 地址更改
- Forged Transits
- Promiscuous Mode Operation

27.7.1.4. 故障切换配置

为了避免停机，可以使用 Deployment 资源部署出口路由器 pod，如下例所示。要为示例部署创建新 Service 对象，使用 oc expose deployment/egress-demo-controller 命令。

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: egress-demo-controller
spec:
  replicas: 1 (1)
  selector:
    matchLabels:
      name: egress-router
  template:
    metadata:
      name: egress-router
      labels:
        name: egress-router
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec: (2)
      initContainers:
        ...
      containers:
        ...
```

(1) 确保副本数被设置为 1，因为在任何同一个时间点上，只有一个 pod 可以使用给定的出口源 IP 地址。这意味着，在一个节点上运行的路由器只有一个副本。

(2) 为出口路由器 pod 指定 **Pod** 对象模板。

27.7.2. 其他资源

- 在重定向模式中部署出口路由器
- 以 HTTP 代理模式部署出口路由器
- 以 DNS 代理模式部署出口路由器

27.8. 以重定向模式部署出口路由器 POD

作为集群管理员，您可以部署一个出口路由器 pod，该 pod 被配置为将流量重新定向到指定的目的地 IP 地址。

27.8.1. 重定向模式的出口路由器 pod 规格

为 **Pod** 对象中的一个出口路由器 pod 定义其配置。以下 YAML 描述了以重定向模式配置出口路由器 pod 的字段：

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
```

```
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress_router>
    - name: EGRESS_GATEWAY (3)
      value: <egress_gateway>
    - name: EGRESS_DESTINATION (4)
      value: <egress_destination>
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod
```

(1) 该注解告知 OpenShift Container Platform 在主网络接口控制器(NIC)上创建 macvlan 网络接口，并将 macvlan 接口移到 pod 的网络命名空间。您必须把 "true" 值包括在引号中。要让 OpenShift Container Platform 在不同的 NIC 接口上创建 macvlan 接口，请将注解值设置为该接口的名称。例如： eth1。

(2) 保留给出口路由器 pod 使用的物理网络的 IP 地址。可选：您可以包括子网长度（/24 后缀），以便正确路由到本地子网。如果没有指定子网长度，则出口路由器只能访问使用 EGRESS_GATEWAY 变量指定的主机，且子网上没有其他主机。

(3) 值与节点使用的默认网关相同。

(4) 将流量定向到的外部服务器。使用这个示例，连接到 pod 流量被重新定向到 203.0.113.25，源 IP 地址为 192.168.12.99。

出口路由器 pod 规格示例

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION
      value: |
        80   tcp 203.0.113.25
        8080 tcp 203.0.113.26 80
```

```
      8443 tcp 203.0.113.26 443
      203.0.113.27
  - name: EGRESS_ROUTER_MODE
    value: init
containers:
- name: egress-router-wait
  image: registry.redhat.io/openshift4/ose-pod
```

27.8.2. 出口目的地配置格式

当出口路由器 pod 被部署为重定向模式时，您可以使用以下一种或多种格式指定重定向规则：

- <port> <protocol> <ip_address> - 到给定 <port> 的内向连接应该被重新定向到给定 <ip_address> 上的同一端口。<protocol> 可以是 tcp 或 udp。
- <port> <protocol> <ip_address> <remote_port> - 和以上一样，除了连接被重新定向到 <ip_address> 上的一个不同的 <remote_port> 中。
- <ip_address> - 如果最后一行是一个 IP 地址，那么其它端口上的所有连接都会被重新指向那个 IP 地址的对应端口。如果没有故障切换 IP 地址，则其它端口上的连接将被拒绝。

在示例中定义了几个规则：

- 第一行将流量从本地端口 80 重定向到 203.0.113.25 的端口 80。
- 第二行和第三行将本地端口 8080 和 8443 重定向到 203.0.113.26 的远程端口 80 和 443。
- 最后一行与之前规则中没有指定的端口的流量匹配。

配置示例

```
80   tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

27.8.3. 以重定向模式部署出口路由器 pod

在*重定向模式*中，出口路由器 pod 会设置 iptables 规则将流量从其自身 IP 地址重定向到一个或多个目标 IP 地址。需要使用保留源 IP 地址的客户端 pod 必须配置为访问出口路由器的服务，而不是直接连接到目标 IP。您可以使用 curl 命令从应用程序 pod 访问目标服务和端口。例如：

```
$ curl <router_service_IP> <port>
```

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建出口路由器 pod。

2. 为确保其他 pod 可以查找出口路由器 pod 的 IP 地址，请创建一个服务指向出口路由器 pod，如下例所示：

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

您的 pod 现在可以连接到此服务。使用保留的出口 IP 地址将其连接重新指向外部服务器的对应端口。

27.8.4. 其他资源

- 使用 ConfigMap 配置出口路由器目的地映射

27.9. 以 HTTP 代理模式部署出口路由器 POD

作为集群管理员，您可以将出口路由器 pod 配置为代理流量到指定的 HTTP 和基于 HTTPS 的服务。

27.9.1. HTTP 模式的出口路由器 pod 规格

为 Pod 对象中的一个出口路由器 pod 定义其配置。以下 YAML 描述了以 HTTP 模式配置出口路由器 pod 的字段：

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress-router>
    - name: EGRESS_GATEWAY (3)
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: http-proxy
```

```
containers:
- name: egress-router-pod
  image: registry.redhat.io/openshift4/ose-egress-http-proxy
  env:
  - name: EGRESS_HTTP_PROXY_DESTINATION 4
    value: |-
      ...
  ...
```

1 该注解告知 OpenShift Container Platform 在主网络接口控制器(NIC)上创建 macvlan 网络接口，并将 macvlan 接口移到 pod 的网络命名空间。您必须把 "true" 值包括在引号中。要让 OpenShift Container Platform 在不同的 NIC 接口上创建 macvlan 接口，请将注解值设置为该接口的名称。例如： eth1。

2 保留给出口路由器 pod 使用的物理网络的 IP 地址。可选：您可以包括子网长度（/24 后缀），以便正确路由到本地子网。如果没有指定子网长度，则出口路由器只能访问使用 EGRESS_GATEWAY 变量指定的主机，且子网上没有其他主机。

3 值与节点使用的默认网关相同。

4 一个字符串或 YAML 多行字符串指定如何配置代理。请注意，这作为 HTTP 代理容器中的环境变量指定，而不是与 init 容器中的其他环境变量指定。

27.9.2. 出口目的地配置格式

当出口路由器 pod 以 HTTP 代理模式部署时，您可以使用以下一个或多个格式指定重定向规则。配置中的每行都指定允许或者拒绝的连接组：

- IP 地址允许连接到那个 IP 地址，如 192.168.1.1。
- CIDR 范围允许连接到那个 CIDR 范围，如 192.168.1.0/24。
- 主机名允许代理该主机，如 www.example.com。
- 前面带有 * 的域名允许代理到那个域及其所有子域，如 *.example.com。
- ! 再加上以前匹配的表达式会拒绝连接。
- 如果最后一行是 *，则任何没有被显式拒绝的都会被允许。否则，任何没有被允许的都会被拒绝。

您还可以使用 * 允许到所有远程目的地的连接。

配置示例

```
!*.example.com
!192.168.1.0/24
192.168.2.1
*
```

27.9.3. 以 HTTP 代理模式部署出口路由器 pod

在 *HTTP 代理模式* 中，出口路由器 pod 作为一个 HTTP 代理在端口8080 上运行。这个模式只适用于连接到基于 HTTP 或基于 HTTPS 服务的客户端，但通常需要较少的更改就可以使客户端 pod 正常工作。很多程序可以通过设置环境变量来使用 HTTP 代理服务器。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建出口路由器 pod。
2. 为确保其他 pod 可以查找出口路由器 pod 的 IP 地址，请创建一个服务指向出口路由器 pod，如下例所示：

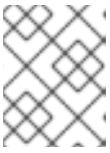
```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http-proxy
    port: 8080 (1)
  type: ClusterIP
  selector:
    name: egress-1
```

(1) 确定 http 端口被设置为 8080。

3. 要将客户端 pod（不是出口代理 Pod）配置为使用 HTTP 代理，设置 http_proxy 或 https_proxy 变量：

```
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
    - name: http_proxy
      value: http://egress-1:8080/ (1)
    - name: https_proxy
      value: http://egress-1:8080/
    ...
```

(1) 上一步中创建的服务。



注意

不需要在所有设置中都使用 http_proxy 和 https_proxy 环境变量。如果以上内容没有创建可以正常工作设置，请查阅 pod 中运行的工具或软件的文档。

27.9.4. 其他资源

- 使用 ConfigMap 配置出口路由器目的地映射

27.10. 以 DNS 代理模式部署出口路由器 POD

作为集群管理员，您可以将配置为代理流量的出口路由器 pod 部署到指定的 DNS 名称和 IP 地址。

27.10.1. DNS 模式的出口路由器 pod 规格

为 Pod 对象中的一个出口路由器 pod 定义其配置。以下 YAML 描述了在 DNS 模式中配置出口路由器 pod 的字段：

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress-router>
    - name: EGRESS_GATEWAY (3)
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
    - name: EGRESS_DNS_PROXY_DESTINATION (4)
      value: |-
        ...
    - name: EGRESS_DNS_PROXY_DEBUG (5)
      value: "1"
    ...
```

(1) 该注解告知 OpenShift Container Platform 在主网络接口控制器(NIC)上创建 macvlan 网络接口，并将 macvlan 接口移到 pod 的网络命名空间。您必须把 "true" 值包括在引号中。要让 OpenShift Container Platform 在不同的 NIC 接口上创建 macvlan 接口，请将注解值设置为该接口的名称。例如： eth1。

(2) 保留给出口路由器 pod 使用的物理网络的 IP 地址。可选：您可以包括子网长度（/24 后缀），以便正确路由到本地子网。如果没有指定子网长度，则出口路由器只能访问使用 EGRESS_GATEWAY 变量指定的主机，且子网上没有其他主机。

3 值与节点使用的默认网关相同。

4 指定一个或多个代理目的地列表。

5 可选：指定输出 DNS 代理日志输出到 stdout。

27.10.2. 出口目的地配置格式

当路由器以 DNS 代理模式部署时，您会指定一个端口和目标映射列表。目的地可以是 IP 地址，也可以是 DNS 名称。

出口路由器 pod 支持以下格式来指定端口和目的地映射：

端口和远程地址

您可以使用两个字段格式来指定源端口和目标主机：<port> <remote_address>。

主机可以是 IP 地址或 DNS 名称。如果提供了 DNS 名称，DNS 解析会在运行时进行。对于给定主机，代理在连接到目标主机的 IP 地址时连接到目标主机上指定的源端口。

端口和远程地址对示例

```
80 172.16.12.11
100 example.com
```

端口、远程地址和远程端口

您可以使用三字段格式 <port> <remote_address> <remote_port> 指定源端口、目标主机和目的地端口。

三字段格式的行为与两字段版本相同，但目的地端口可能与源端口不同。

端口、远程地址和远程端口示例

```
8080 192.168.60.252 80
8443 web.example.com 443
```

27.10.3. 以 DNS 代理模式部署出口路由器 pod

在 *DNS 代理模式* 中，出口路由器 pod 作为基于 TCP 服务的 DNS 代理运行，将其自身的 IP 地址转换到一个或多个目标 IP 地址。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建出口路由器 pod。
2. 为出口路由器 pod 创建服务：

a. 创建名为 **egress-router-service.yaml** 的文件，其包含以下 YAML。将 **spec.ports** 设置为您之前为 **EGRESS_DNS_PROXY_DESTINATION** 环境变量定义的端口列表。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    ...
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

例如：

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
  - name: con1
    protocol: TCP
    port: 80
    targetPort: 80
  - name: con2
    protocol: TCP
    port: 100
    targetPort: 100
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

b. 要创建服务，请输入以下命令：

```
$ oc create -f egress-router-service.yaml
```

Pod 现在可以连接至此服务。使用保留的出口 IP 地址将其代理到外部服务器的对应端口。

27.10.4. 其他资源

- 使用 ConfigMap 配置出口路由器目的地映射

27.11. 从配置映射配置出口路由器 POD 目的地列表

作为集群管理员，您可以定义一个 **ConfigMap** 对象来指定出口路由器 pod 的目标映射。配置的特定格式取决于出口路由器 pod 的类型。有关格式的详情，请参阅特定出口路由器 pod 的文档。

27.11.1. 使用配置映射配置出口路由器目的地映射

对于大量或经常更换的目标映射集合，您可以使用配置映射来外部维护列表。这种方法的优点是可将编辑配置映射的权限委派给没有 **cluster-admin** 特权的用户。因为出口路由器 pod 需要特权容器，没有 **cluster-admin** 特权的用户无法直接编辑 pod 定义。

注意

配置映射更改时，出口路由器 pod 不会自动更新。您必须重启出口路由器 pod 来获得更新。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**

流程

1. **创建包含出口路由器 pod 映射数据的文件，如下例所示：**

```
# Egress routes for Project "Test", version 3

80   tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

您可以在这个文件中放入空白行和评论。

2. **从文件创建 ConfigMap 对象：**

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

在以前的版本中，egress-routes 值是要创建的 ConfigMap 对象的名称，my-egress-destination.txt 是数据从中读取的文件的名称。

提示

您还可以应用以下 YAML 来创建配置映射：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: egress-routes
data:
  destination: |
    # Egress routes for Project "Test", version 3

    80   tcp 203.0.113.25

    8080 tcp 203.0.113.26 80
    8443 tcp 203.0.113.26 443

    # Fallback
    203.0.113.27
```

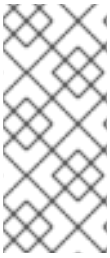
3. 创建出口路由器 pod 定义，并为环境片段中的 EGRESS_DESTINATION 字段指定 configMapKeyRef 小节：

```
...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...
```

27.11.2. 其他资源

- 重定向模式
- HTTP 代理模式
- DNS 代理模式

27.12. 为项目启用多播



注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

27.12.1. 关于多播

通过使用 IP 多播，数据可同时广播到许多 IP 地址。

重要

- 目前，多播最适用于低带宽协调或服务发现。它不是一个高带宽解决方案。
- 默认情况下，网络策略会影响命名空间中的所有连接。但是，多播不受网络策略的影响。如果在与网络策略相同的命名空间中启用了多播，则始终允许多播，即使有一个 deny-all 网络策略。在启用网络策略前，集群管理员应考虑对多播的影响。

默认情况下，OpenShift Container Platform pod 之间多播流量被禁用。如果使用 OpenShift SDN 网络插件，可以根据每个项目启用多播。

在 networkpolicy 隔离模式中使用 OpenShift SDN 网络插件：

- pod 发送的多播数据包将传送到项目中的所有其他 pod，而无需考虑 NetworkPolicy 对象。即使在无法通过单播通信时，Pod 也能通过多播进行通信。
- 一个项目中的 pod 发送的多播数据包不会传送到任何其他项目中的 pod，即使存在允许项目间通信的 NetworkPolicy 对象。

以 multitenant 隔离模式使用 OpenShift SDN 网络插件时：

- pod 发送的多播数据包将传送到项目中的所有其他 pod。
- 只有在各个项目接合在一起并且每个接合的项目上都启用了多播时，一个项目中的 pod 发送的多播数据包才会传送到其他项目中的 pod。

27.12.2. 启用 pod 间多播

您可以为项目启用 pod 间多播。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 运行以下命令，为项目启用多播。使用您要启用多播的项目的名称替换 <namespace>。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

验证

要验证项目是否启用了多播，请完成以下步骤：

1. 将您的当前项目更改为启用多播的项目。使用项目名替换 <project>。

```
$ oc project <project>
```

2. 创建 pod 以作为多播接收器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
```

```
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. 创建 pod 以作为多播发送器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 在新的终端窗口或选项卡中，启动多播监听程序。

   a. 获得 Pod 的 IP 地址：

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. 输入以下命令启动多播监听程序：

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. 启动多播传输。

   a. 获取 pod 网络 IP 地址范围：

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

b. 要发送多播信息，请输入以下命令：

```
$ oc exec msender -i -t -- \
    /bin/bash -c "echo | socat STDIO UDP4-
DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

如果多播正在工作，则上一个命令会返回以下输出：

```
mlistener
```

27.13. 为项目禁用多播

27.13.1. 禁用 pod 间多播

您可以为项目禁用 pod 间多播。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 运行以下命令来禁用多播：

```
$ oc annotate netnamespace <namespace> \ 1
    netnamespace.network.openshift.io/multicast-enabled-
```

1 您要禁用多播的项目的 namespace。

27.14. 使用 OPENSHIFT SDN 配置网络隔离

注意

从 OpenShift Container Platform 4.14 开始，OpenShift SDN CNI 已被弃用。自 OpenShift Container Platform 4.15 起，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

当集群配置为使用 OpenShift SDN 网络插件的多租户隔离模式时，每个项目会被默认隔离。在多租户隔离模式下，不同项目中的 pod 或服务间不允许网络流量。

您可以通过两种方式更改项目的多租户隔离行为：

- 您可以接合一个或多个项目，允许不同项目中的 pod 和服务间的网络流量。
- 您可以对项目禁用网络隔离。它可全局访问，接受所有其他项目中的 pod 和服务的网络流量。可全局访问的项目可以访问所有其他项目中的 pod 和服务。

27.14.1. 先决条件

- 您必须将集群配置为在多租户隔离模式中使用 OpenShift SDN 网络插件。

27.14.2. 接合项目

您可以接合两个或多个项目，以允许不同项目中的 Pod 和服务间的网络流量。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

1. 使用以下命令，将项目接合到现有项目网络中：

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

另外，除了指定具体的项目名称，也可以使用 --selector=<project_selector> 选项来基于关联标签指定项目。

2. 可选：运行以下命令来查看您接合在一起的 Pod 网络：

```
$ oc get netnamespaces
```

在 NETID 列中，同一 Pod 网络中的项目具有相同的网络 ID。

27.14.3. 隔离项目

您可以隔离项目，使其他项目中的 pod 和服务无法访问这个项目中的 pod 和服务。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 要隔离集群中的项目，请运行以下命令：

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

另外，除了指定具体的项目名称，也可以使用 --selector=<project_selector> 选项来基于关联标签指定项目。

27.14.4. 对项目禁用网络隔离

您可以对项目禁用网络隔离。

先决条件

- 安装 OpenShift CLI（oc）。
- 您必须作为 cluster-admin 角色用户登录集群。

流程

- 对项目运行以下命令：

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

另外，除了指定具体的项目名称，也可以使用 --selector=<project_selector> 选项来基于关联标签指定项目。

27.15. 配置 KUBE-PROXY

Kubernetes 网络代理 (kube-proxy) 在每个节点上运行，并由 Cluster Network Operator (CNO) 管理。kube-proxy 维护网络规则，以转发与服务关联的端点的连接。

27.15.1. 关于 iptables 规则同步

同步周期决定 Kubernetes 网络代理 (kube-proxy) 在节点上同步 iptables 规则的频率。

同步在发生以下事件之一时开始：

- 发生某一事件，例如服务或端点添加到集群中或从集群中删除。
- 距最后一次同步的时间已超过为 kube-proxy 定义的同步周期。

27.15.2. kube-proxy 配置参数

您可以修改以下 kubeProxyConfig 参数。

注意

由于 OpenShift Container Platform 4.3 及更高版本中引进了性能上的改进，现在不再需要调整 iptablesSyncPeriod 参数。

表 27.2. 参数

参数	描述	值	默认值
iptablesSyncPeriod	iptables 规则的刷新周期。	一个时间间隔，如 30s 或 2m。有效的后缀包括 s、m和 h，具体参见 Go 时间包文档。	30s
proxyArguments.iptables-min-sync-period	刷新 iptables 规则前的最短时长。此参数确保刷新的频率不会过于频繁。默认情况下，当发生影响 iptables 规则的更改时就会立即进行刷新。	一个时间间隔，如 30s 或 2m。有效的后缀包括 s、m和 h，具体参见 Go 时间包	0s

27.15.3. 修改 kube-proxy 配置

您可以为集群修改 Kubernetes 网络代理配置。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用 cluster-admin 角色登录正在运行的集群。

流程

1. 运行以下命令来编辑 Network.operator.openshift.io 自定义资源（CR）：

```
$ oc edit network.operator.openshift.io cluster
```

2. 利用您对 kube-proxy 配置的更改修改 CR 中的 kubeProxyConfig 参数，如以下示例 CR 中所示：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period: ["30s"]
```

3. 保存文件并退出文本编辑器。
保存文件并退出编辑器时，oc 命令会验证其语法。如果您的修改含有语法错误，编辑器会打开该文件并显示错误消息。

4. 运行以下命令来确认配置更新：

```
$ oc get networks.operator.openshift.io -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
        - 30s
```

```
    serviceNetwork:
    - 172.30.0.0/16
  status: {}
kind: List
```

5. 可选：运行以下命令，确认 Cluster Network Operator 已接受配置更改：

```
$ oc get clusteroperator network
```

输出示例

```
NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.1.0-0.9   True        False         False      1m
```

成功应用配置更新后，AVAILABLE 字段为 True。

第 28 章 配置路由

28.1. 路由配置

28.1.1. 创建基于 HTTP 的路由

路由允许您在公共 URL 托管应用程序。根据应用程序的网络安全配置，它可以安全或不受保护。基于 HTTP 的路由是一个不受保护的路由，它使用基本的 HTTP 路由协议，并在未安全的应用程序端口上公开服务。

以下流程描述了如何使用 hello-openshift 应用程序创建基于 HTTP 的简单路由，作为示例。

前提条件

- 已安装 OpenShift CLI（oc）。
- 以管理员身份登录。
- 您有一个 web 应用，用于公开端口和侦听端口上流量的 TCP 端点。

流程

1. 运行以下命令，创建一个名为 hello-openshift 的项目：

```
$ oc new-project hello-openshift
```

2. 运行以下命令，在项目中创建 pod：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-
openshift/hello-pod.json
```

3. 运行以下命令，创建名为 hello-openshift 的服务：

```
$ oc expose pod/hello-openshift
```

4. 运行以下命令，创建一个没有安全安全的路由到 hello-openshift 应用程序：

```
$ oc expose svc hello-openshift
```

验证

- 要验证您创建的路由资源，请运行以下命令：

```
$ oc get routes -o yaml <name of resource>
```
(1)

(1) 在本例中，路由名为 hello-openshift。

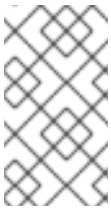
创建的未安全路由的 YAML 定义示例：

```
apiVersion: route.openshift.io/v1
```

```
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain> 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

1 <Ingress_Domain> 是默认的入口域名。ingresses.config/cluster 对象是在安装过程中创建的，且无法更改。如果要指定不同的域，您可以使用 appsDomain 选项指定备选集群域。

2 targetPort 是由此路由指向的服务选择的 pod 上的目标端口。



注意

要显示您的默认入口域，请运行以下命令：

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

28.1.2. 为 Ingress Controller 分片创建路由

通过使用路由，您可以通过 URL 托管应用程序。在这种情况下，主机名没有被设置，路由会使用子域。当您指定子域时，会自动使用公开路由的 Ingress Controller 域。对于由多个 Ingress Controller 公开路由的情况，路由由多个 URL 托管。

以下流程描述了如何为 Ingress Controller 分片创建路由，使用 hello-openshift 应用程序作为示例。

在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您以项目管理员身份登录。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。
- 您已为分片配置了 Ingress Controller。

流程

1. 运行以下命令，创建一个名为 hello-openshift 的项目：

```
$ oc new-project hello-openshift
```

2. 运行以下命令，在项目中创建 pod：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-
openshift/hello-pod.json
```

3. 运行以下命令，创建名为 hello-openshift 的服务：

```
$ oc expose pod/hello-openshift
```

4. 创建名为 hello-openshift-route.yaml 的路由定义：

   为分片创建的路由的 YAML 定义：

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

   (1) 标签键及其对应标签值必须与 Ingress Controller 中指定的标签值匹配。在本例中，Ingress Controller 具有标签键和值 type: sharded。

   (2) 路由将使用 subdomain 字段的值公开。指定 subdomain 字段时，您必须保留主机名未设置。如果您同时指定了 host 和 subdomain 字段，则路由将使用 host 字段的值，并忽略 subdomain 字段。

5. 通过运行以下命令，使用 hello-openshift-route.yaml 创建到 hello-openshift 应用程序的路由：

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

验证

- 使用以下命令获取路由的状态：

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

   生成的 Route 资源应类似以下示例：

   输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
```

```
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-
sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1 Ingress Controller 或路由器的主机名用于公开路由。host 字段的值由 Ingress Controller 自动决定，并使用它的域。在本例中，Ingress Controller 的域为 <apps-sharded.basedomain.example.net>。

2 Ingress Controller 的主机名。

3 Ingress Controller 的名称。在本例中，Ingress Controller 的名称为 sharded。

28.1.3. 配置路由超时

如果您的服务需要低超时（满足服务级别可用性 (SLA) 目的）或高超时（具有慢速后端的情况），您可以为现有路由配置默认超时。

前提条件

- 您需要在运行的集群中部署了 Ingress Controller。

流程

1. 使用 oc annotate 命令，为路由添加超时：

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1 支持的时间单位是微秒 (us)、毫秒 (ms)、秒钟 (s)、分钟 (m)、小时 (h)、或天 (d)。

以下示例在名为 myroute 的路由上设置两秒的超时：

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

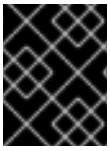
28.1.4. HTTP 严格传输安全性

HTTP 严格传输安全性 (HSTS) 策略是一种安全增强，向浏览器客户端发送信号，表示路由主机上仅允许 HTTPS 流量。HSTS 也通过信号 HTTPS 传输来优化 Web 流量，无需使用 HTTP 重定向。HSTS 对于加快与网站的交互非常有用。

强制 HSTS 策略时，HSTS 会向站点的 HTTP 和 HTTPS 响应添加 Strict Transport Security 标头。您可以在路由中使用 insecureEdgeTerminationPolicy 值，以将 HTTP 重定向到 HTTPS。强制 HSTS 时，客户端会在发送请求前将所有请求从 HTTP URL 更改为 HTTPS，无需重定向。

集群管理员可将 HSTS 配置为执行以下操作：

- 根据每个路由启用 HSTS
- 根据每个路由禁用 HSTS
- 对一组域强制每个域的 HSTS，或者结合使用命名空间标签与域



重要

HSTS 仅适用于安全路由，可以是 edge-terminated 或 re-encrypt。其配置在 HTTP 或传递路由上无效。

28.1.4.1. 根据每个路由启用 HTTP 严格传输安全性

HTTP 严格传输安全 (HSTS) 实施在 HAProxy 模板中，并应用到具有 haproxy.router.openshift.io/hsts_header 注解的边缘和重新加密路由。

前提条件

- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 OpenShift CLI（oc）。

流程

- 要在路由上启用 HSTS，请将 haproxy.router.openshift.io/hsts_header 值添加到 edge-terminated 或 re-encrypt 路由中。您可以运行以下命令来使用 oc annotate 工具来实现此目的：

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000;\ 1
includeSubDomains;preload"
```

1 在本例中，最长期限设置为 31536000 ms，大约为 8.5 小时。

注意

在这个示例中，等号 (=) 包括在引号里。这是正确执行注解命令所必需的。

配置了注解的路由示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-
age=31536000;includeSubDomains;preload 1 2 3
...
```

```
spec:
  host: def.abc.com
  tls:
    termination: "reencrypt"
    ...
  wildcardPolicy: "Subdomain"
```

1. 必需。Max-age 测量 HSTS 策略生效的时间长度，以秒为单位。如果设置为0，它将对策略进行求反。
2. 可选。包含时，includeSubDomains 告知客户端主机的所有子域都必须与主机具有相同的 HSTS 策略。
3. 可选。当 max-age 大于 0 时，您可以在haproxy.router.openshift.io/hsts_header 中添加 preload，以允许外部服务将这个站点包括在 HSTS 预加载列表中。例如，Google 等站点可以构造设有 preload 的站点的列表。浏览器可以使用这些列表来确定哪些站点可以通过 HTTPS 通信，即使它们与站点交互之前也是如此。如果没有设置 preload，浏览器必须已经通过 HTTPS 与站点交互（至少一次）才能获取标头。

28.1.4.2. 根据每个路由禁用 HTTP 严格传输安全性

要禁用 HTTP 严格传输安全性 (HSTS)，您可以将路由注解中的 max-age 值设置为 0。

前提条件

- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 OpenShift CLI（oc）。

流程

- 要禁用 HSTS，请输入以下命令将路由注解中的 max-age 值设置为 0 ：

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

提示

您还可以应用以下 YAML 来创建配置映射：

根据每个路由禁用 HSTS 的示例

```
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- 要为命名空间中的所有路由禁用 HSTS，请输入以下命令：

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

验证

1. 要查询所有路由的注解，请输入以下命令：

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if
.metadata.annotations}}{{$a := index .metadata.annotations
"haproxy.router.openshift.io/hsts_header"}}{{$n := .metadata.name}}{{with $a}}Name:
{{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
```

输出示例

```
Name: routename HSTS: max-age=0
```

28.1.4.3. 强制每个域 HTTP 严格传输安全性

要为安全路由强制实施 HTTP Strict Transport Security (HSTS)，在 Ingress spec 中添加 requiredHSTSPolicies 记录来捕获 HSTS 策略的配置。

如果您将 requiredHSTSPolicy 配置为强制 HSTS，则任何新创建的路由都必须配置有兼容的 HSTS 策略注解。

注意

要使用不合规的 HSTS 路由处理升级的集群，您可以在源更新清单并应用更新。

注意

您无法使用 oc expose route 或 oc create route 命令在强制 HSTS 的域中添加路由，因为这些命令的 API 不接受注解。

重要

HSTS 无法应用到不安全或非 TLS 路由，即使 HSTS 全局请求了 HSTS。

先决条件

- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 OpenShift CLI（oc）。

流程

1. 运行以下命令来编辑 Ingress 配置 YAML，并根据需要更新字段：

```
$ oc edit ingresses.config.openshift.io/cluster
```

HSTS 策略示例

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: 'hello-openshift-default.apps.username.devcluster.openshift.com'
```

```
requiredHSTSPolicies: (1)
- domainPatterns: (2)
  - '*hello-openshift-default.apps.username.devcluster.openshift.com'
  - '*hello-openshift-default2.apps.username.devcluster.openshift.com'
  namespaceSelector: (3)
    matchLabels:
      myPolicy: strict
  maxAge: (4)
    smallestMaxAge: 1
    largestMaxAge: 31536000
  preloadPolicy: RequirePreload (5)
  includeSubDomainsPolicy: RequireIncludeSubDomains (6)
- domainPatterns:
  - 'abc.example.com'
  - '*xyz.example.com'
  namespaceSelector:
    matchLabels: {}
  maxAge: {}
  preloadPolicy: NoOpinion
  includeSubDomainsPolicy: RequireNoIncludeSubDomains
```

(1) 必需。requiredHSTSPolicies 会被按顺序验证，并应用第一个匹配的 domainPatterns。

(2) 必需。您必须至少指定一个 domainPatterns 主机名。可以列出任意数量的域。您可以为不同的 domainPatterns 包括多个强制选项部分。

(3) 可选。如果包含 namespaceSelector，它必须与路由所在项目的标签匹配，以便在路由上强制执行设定 HSTS 策略。仅与 namespaceSelector 而不是 domainPatterns 匹配的路由不会被验证。

(4) 必需。Max-age 测量 HSTS 策略生效的时间长度，以秒为单位。此策略设置允许强制实施最小和最大的 max-age。

- largestMaxAge 值必须在 0 到 2147483647 之间。它可以不指定，这意味着不强制实施上限。
- smallestMaxAge 值必须在 0 到 2147483647 之间。输入 0 来禁用 HSTS 以进行故障排除，或者如果您不需要禁用 HSTS，输入 1。它可以不知道，这意味着不强制实施较低限制。

(5) 可选。在 haproxy.router.openshift.io/hsts_header 中包含 preload 会使外部服务将此站点包括在 HSTS 预加载列表中。浏览器可以使用这些列表来决定哪些站点可通过 HTTPS 进行通信，然后再与站点交互。如果没有设置 preload，浏览器需要至少与站点交互一次，才能获取该标头。可使用以下方法之一设置 preload ：

- RequirePreload：RequiredHSTSPolicy 需要 preload。
- RequireNoPreload:preload 被 RequiredHSTSPolicy 禁止。
- NoOpinion：preload 与 RequiredHSTSPolicy 没有关系。

(6) 可选。includeSubDomainsPolicy 可使用以下之一设置：

- RequireIncludeSubDomains: RequiredHSTSPolicy 需要 includeSubDomains。

- RequireNoIncludeSubDomains: includeSubDomains 被 RequiredHSTSPolicy 禁止。
- NoOpinion : includeSubDomains 与 RequiredHSTSPolicy 没有关系。

2. 您可以通过输入 oc annotate command，将 HSTS 应用到集群或特定命名空间中的所有路由。
   - 要将 HSTS 应用到集群中的所有路由，请输入 oc annotate command。例如：

     ```
     $ oc annotate route --all --all-namespaces --overwrite=true
     "haproxy.router.openshift.io/hsts_header"="max-age=31536000"
     ```

   - 要将 HSTS 应用到特定命名空间中的所有路由，请输入 oc annotate command。例如：

     ```
     $ oc annotate route --all -n my-namespace --overwrite=true
     "haproxy.router.openshift.io/hsts_header"="max-age=31536000"
     ```

验证

您可以查看您配置的 HSTS 策略。例如：

- 要查看所需的 HSTS 策略的 maxAge 设置，请输入以下命令：

  ```
  $ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range
  .spec.requiredHSTSPolicies[*]}
  {.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge}{"\n"}{end}'
  ```

- 要查看所有路由上的 HSTS 注解，请输入以下命令：

  ```
  $ oc get route  --all-namespaces -o go-template='{{range .items}}{{if
  .metadata.annotations}}{{$a := index .metadata.annotations
  "haproxy.router.openshift.io/hsts_header"}}{{$n := .metadata.name}}{{with $a}}Name:
  {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
  ```

  输出示例

  ```
  Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
  ```

28.1.5. 吞吐量问题的故障排除方法

有时，通过 OpenShift Container Platform 部署的应用程序可能会导致网络吞吐量问题，如特定服务间的延迟异常高。

如果 pod 日志没有显示造成问题的原因，请使用以下方法之一分析性能问题：

- 使用 ping 或 tcpdump 等数据包分析器，分析 pod 与其节点之间的流量。
  例如，在每个 pod 上运行 tcpdump 工具，同时重现导致问题的行为。检查两端的捕获信息，以便比较发送和接收时间戳来分析与 pod 往来的流量的延迟。如果节点接口被其他 pod、存储设备或者数据平面的流量过载，则 OpenShift Container Platform 中可能会出现延迟。

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> 1
  ```

1 podip 是 pod 的 IP 地址。运行oc get pod <pod_name> -o wide 命令来获取 pod 的 IP 地址。

tcpdump 命令会在 /tmp/dump.pcap 中生成一个包含这两个 pod 间所有流量的文件。您可以在运行分析器后立即重现问题，并在问题重现完成后马上停止分析器，从而尽量减小文件的大小。您还可以通过以下命令，在节点之间运行数据包分析器（从考量范围中剔除 SDN）：

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- 使用 iperf 等带宽测量工具来测量流吞吐量和 UDP 吞吐量。首先从 pod 运行该工具，然后从节点运行它，从而找到瓶颈。
  - 有关安装和使用 iperf 的详情，请参考此红帽解决方案。
- 在某些情况下，因为延迟问题，集群可能会将带有路由器 pod 的节点标记为不健康。在执行操作前，使用 worker 延迟配置集调整集群等待节点状态更新的频率。
- 如果您的集群指定了较低延迟和较高延迟节点，请将 Ingress Controller 中的 spec.nodePlacement 字段配置为控制路由器 pod 的放置。

其他资源

- 远程 worker 吞吐量延迟或临时减少
- Ingress Controller 配置参数

28.1.6. 使用 Cookie 来保持路由有状态性

OpenShift Container Platform 提供粘性会话，通过确保所有流量都到达同一端点来实现有状态应用程序流量。但是，如果端点 pod 以重启、扩展或更改配置的方式被终止，这种有状态性可能会消失。

OpenShift Container Platform 可以使用 Cookie 来配置会话持久性。ingress 控制器选择一个端点来处理任何用户请求，并为会话创建一个 Cookie。Cookie 在响应请求时返回，用户则通过会话中的下一请求发回 Cookie。Cookie 告知 Ingress Controller 哪个端点正在处理会话，确保客户端请求使用这个 Cookie 使请求路由到同一个 pod。

注意

无法在 passthrough 路由上设置 Cookie，因为无法看到 HTTP 流量。相反，根据源 IP 地址计算数字，该地址决定了后端。

如果后端更改，可以将流量定向到错误的服务器，使其更不计。如果您使用负载均衡器来隐藏源 IP，则会为所有连接和流量都发送到同一 pod 设置相同的数字。

28.1.6.1. 使用 Cookie 标注路由

您可以设置 Cookie 名称来覆盖为路由自动生成的默认名称。这样，接收路由流量的应用程序就能知道 Cookie 名称。通过删除 Cookie，它可以强制下一请求重新选择端点。结果是，如果服务器过载，该服务器会尝试从客户端中删除请求并重新分发它们。

流程

1. 使用指定的 Cookie 名称标注路由：

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

其中：

<route_name>
指定路由的名称。
<cookie_name>
指定 Cookie 的名称。

例如，使用 cookie 名称 my_cookie 标注路由 my_route：

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. 在变量中捕获路由主机名：

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

其中：

<route_name>
指定路由的名称。

3. 保存 cookie，然后访问路由：

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

使用上一个命令在连接到路由时保存的 cookie：

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

28.1.7. 基于路径的路由

基于路径的路由指定了一个路径组件，可以与 URL 进行比较，该 URL 需要基于 HTTP 的路由流量。因此，可以使用同一主机名提供多个路由，每个主机名都有不同的路径。路由器应该匹配基于最具体路径的路由。

下表显示了路由及其可访问性示例：

表 28.1. 路由可用性

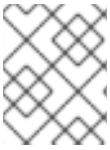
Route（路由）	当比较到	可访问
www.example.com/test	*www.example.com/test*	是
	www.example.com	否
www.example.com/test 和 *www.example.com*	*www.example.com/test*	是
	www.example.com	是

Route（路由）	当比较到	可访问
www.example.com	*www.example.com/text*	yes（由主机匹配，而不是路由）
	www.example.com	是

带有路径的未安全路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1 该路径是基于路径的路由的唯一添加属性。



注意

使用 passthrough TLS 时，基于路径的路由不可用，因为路由器不会在这种情况下终止 TLS，且无法读取请求的内容。

28.1.8. HTTP 标头配置

OpenShift Container Platform 提供了不同的使用 HTTP 标头的方法。在设置或删除标头时，您可以使用 Ingress Controller 中的特定字段或单独的路由来修改请求和响应标头。您还可以使用路由注解设置某些标头。配置标头的各种方法在协同工作时可能会带来挑战。

注意

您只能在 IngressController 或 Route CR 中设置或删除标头，您无法附加它们。如果使用值设置 HTTP 标头，则该值必须已完成，且在以后不需要附加。在附加标头（如 X-Forwarded-For 标头）时，请使用 spec.httpHeaders.forwardedHeaderPolicy 字段，而不是 spec.httpHeaders.actions。

28.1.8.1. 优先级顺序

当在 Ingress Controller 和路由中修改相同的 HTTP 标头时，HAProxy 会根据它是请求还是响应标头来优先选择操作。

- 对于 HTTP 响应标头，Ingress Controller 中指定的操作会在路由中指定的操作后执行。这意味着 Ingress Controller 中指定的操作具有优先权。
- 对于 HTTP 请求标头，路由中指定的操作会在 Ingress Controller 中指定的操作后执行。这意味着路由中指定的操作具有优先权。

例如，集群管理员使用以下配置设置 X-Frame-Options 响应标头，其值为 DENY ：

IngressController spec 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

路由所有者设置 Ingress Controller 中设置的相同响应标头，但使用以下配置值 SAMEORIGIN ：

Route 规格示例

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: SAMEORIGIN
```

当 IngressController spec 和 Route spec 都配置 X-Frame-Options 标头时，Ingress Controller 中的全局级别为这个标头设置的值将具有优先权，即使一个特定的路由允许帧。对于请求标头，Route spec 值会覆盖 IngressController spec 值。

发生这个优先级，因为 haproxy.config 文件使用以下逻辑，其中 Ingress Controller 被视为前端，单独的路由被视为后端。应用到前端配置的标头值 DENY 使用后端中设置的值 SAMEORIGIN 覆盖相同的标头：

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

另外，Ingress Controller 或路由中定义的任何操作都覆盖使用路由注解设置的值。

28.1.8.2. 特殊情况标头

以下标头可能会阻止完全被设置或删除，或者在特定情况下允许：

表 28.2. 特殊情况标头配置选项

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
proxy	否	否	**proxy** HTTP 请求标头可以通过将标头值注入 **HTTP_PROXY** 环境变量来利用这个安全漏洞的 CGI 应用程序。**proxy** HTTP 请求标头也是非标准的，在配置期间容易出错。	否
主机	否	是	当使用 **IngressController** CR 设置 **host** HTTP 请求标头时，HAProxy 在查找正确的路由时可能会失败。	否
strict-transport-security	否	否	**strict-transport-security** HTTP 响应标头已使用路由注解处理，不需要单独的实现。	是：**haproxy.router.openshift.io/hsts_header** 路由注解
cookie 和 **set-cookie**	否	否	HAProxy 集的 Cookie 用于会话跟踪，用于将客户端连接映射到特定的后端服务器。允许设置这些标头可能会影响 HAProxy 的会话关联，并限制 HAProxy 的 Cookie 的所有权。	是： • **haproxy.router.openshift.io/disable_cookie** 路由注解 • **haproxy.router.openshift.io/cookie_name** 路由注解

28.1.9. 在路由中设置或删除 HTTP 请求和响应标头

出于合规的原因，您可以设置或删除某些 HTTP 请求和响应标头。您可以为 Ingress Controller 提供的所有路由或特定路由设置或删除这些标头。

例如，如果内容使用多种语言编写，您可能希望让 Web 应用程序在备用位置提供内容，即使 Ingress Controller 为路由指定的默认全局位置。

以下流程会创建一个设置 Content-Location HTTP 请求标头的路由，以便与应用程序关联的 URL https://app.example.com 定向到位置 https://app.example.com/lang/en-us。将应用程序流量定向到此位置意味着使用该特定路由的任何人都可以访问以美国英语编写的 Web 内容。

先决条件

- 已安装 OpenShift CLI（oc）。
- 以项目管理员身份登录到 OpenShift Container Platform 集群。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。

流程

1. 创建一个路由定义，并将它保存到名为 app-example-route.yaml 的文件中：

   使用 HTTP 标头指令创建路由的 YAML 定义

```yaml
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
    name: app-example
  httpHeaders:
    actions: 1
      response: 2
      - name: Content-Location 3
        action:
          type: Set 4
          set:
            value: /lang/en-us 5
```

1. 要在 HTTP 标头上执行的操作列表。
2. 您要更改的标头类型。在本例中，响应标头。
3. 您要更改的标头的名称。有关您可以设置或删除的可用标头列表，请参阅 *HTTP 标头配置*。
4. 在标头中执行的操作类型。此字段可以具有 Set 或 Delete 的值。
5. 在设置 HTTP 标头时，您必须提供一个 value。该值可以是该标头的可用指令列表中的字符串，如 DENY，也可以是使用 HAProxy 的动态值语法来解释的动态值。在这种情况下，该值被设置为内容的相对位置。

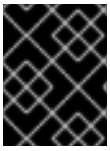
2. 使用新创建的路由定义，创建到现有 Web 应用程序的路由：

```
$ oc -n app-example create -f app-example-route.yaml
```

对于 HTTP 请求标头，路由定义中指定的操作会在 Ingress Controller 中对 HTTP 请求标头执行的任何操作后执行。这意味着，路由中这些请求标头设置的任何值都将优先于 Ingress Controller 中设置的值。有关 HTTP 标头处理顺序的更多信息，请参阅 *HTTP 标头配置*。

28.1.10. 特定于路由的注解

Ingress Controller 可以为它公开的所有路由设置默认选项。单个路由可以通过在其注解中提供特定配置来覆盖这些默认设置。红帽不支持在 Operator 管理的路由中添加路由注解。



重要

要创建带有多个源 IP 或子网的白名单，请使用以空格分隔的列表。任何其他限定类型会导致忽略列表，而不发出警告或错误消息。

表 28.3. 路由注解

变量	描述	默认的环境变量
haproxy.router.openshift.io/balance	设置负载平衡算法。可用选项是 **random**、**source**、**roundrobin** 和 **leastconn**。对于 TLS 透传路由，默认值为 **source**。对于所有其他路由，默认值为 **random**。	passthrough 路由 使用 **ROUTER_TCP_BALANCE_SCHEME** 。否则，使用 **ROUTER_LOAD_BALANCE_algorithm**。
haproxy.router.openshift.io/disable_cookies	禁用使用 cookie 来跟踪相关连接。如果设置为 **'true'** 或 **'TRUE'**，则使用均衡算法选择每个传入 HTTP 请求的后端服务连接。	
router.openshift.io/cookie_name	指定一个可选的、用于此路由的 cookie。名称只能包含大写字母和小写字母、数字、"_" 和 "-"。默认为路由的内部密钥进行哈希处理。	
haproxy.router.openshift.io/pod-concurrent-connections	设置路由器支持的 pod 允许的最大连接数。 注： 如果有多个 pod，每个 pod 都有这些数量的连接。如果有多个路由器，它们之间没有协调关系，每个路由器都可能会多次连接。如果没有设置，或者将其设定为 0，则没有限制。	

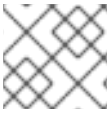
变量	描述	默认的环境变量
haproxy.router.openshift.io/rate-limit-connections	设置 **'true'** 或 **'TRUE'** 可启用速率限制功能，该功能通过每个路由上的特定后端的贴子实施。 注： 使用此注解提供了对拒绝服务攻击的基本保护。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	限制通过同一源 IP 地址进行的并发 TCP 连接数。它接受一个数字值。 注： 使用此注解提供了对拒绝服务攻击的基本保护。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	限制具有相同源 IP 地址的客户端可以发出 HTTP 请求的速率。它接受一个数字值。 注： 使用此注解提供了对拒绝服务攻击的基本保护。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	限制具有相同源 IP 地址的客户端可以进行 TCP 连接的速率。它接受一个数字值。 注： 使用此注解提供了对拒绝服务攻击的基本保护。	
haproxy.router.openshift.io/timeout	为路由设定服务器端超时。(TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
haproxy.router.openshift.io/timeout-tunnel	这个超时适用于隧道连接，如明文、边缘、重新加密或透传路由。使用明文、边缘或重新加密路由类型，此注解作为带有现有超时值的超时隧道应用。对于 passthrough 路由类型，注解优先于设置任何现有的超时值。	**ROUTER_DEFAULT_TUNNEL_TIMEOUT**
ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after	您可以设置 IngressController 或 ingress 配置。此注解重新部署路由器，并将 HA 代理配置为在全局后发出 haproxy **hard-stop- after** 全局 选项，用于定义执行干净的软停止的最长时间。	**ROUTER_HARD_STOP_AFTER**
router.openshift.io/haproxy.health.check.interval	为后端健康检查设定间隔。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**

变量	描述	默认的环境变量
haproxy.router.openshift.io/ip_whitelist	为路由设置允许列表。允许列表（allowlist）是以空格分开的 IP 地址和 CIDR 范围列表，用来代表批准的源地址。不是来自允许列表中的 IP 地址的请求会被丢弃。 在 **haproxy.config** 文件中直接可见的最大 IP 地址和 CIDR 范围数为 61. [1]	
haproxy.router.openshift.io/hsts_header	为 edge terminated 或 re-encrypt 路由设置 Strict-Transport-Security 标头。	
haproxy.router.openshift.io/rewrite-target	在后端中设置请求的重写路径。	
router.openshift.io/cookie-same-site	设置一个值来限制 cookies。数值是： **Lax**: 浏览器不会在跨站点请求上发送 Cookie，当用户从外部站点导航到原始站点时发送 Cookie。当未指定 **SameSite** 值时，这是默认的浏览器行为。 **Strict** ：浏览器仅针对同一站点请求发送 Cookie。 **None** ：浏览器为跨站点和相同站点请求发送 Cookie。 这个值仅适用于重新加密和边缘路由。如需更多信息，请参阅 SameSite cookies 文档。	
haproxy.router.openshift.io/set-forwarded-headers	设置用于处理每个路由的 **Forwarded** 和 **X-Forwarded-For** HTTP 标头的策略。数值是： **Append**附加标头，保留任何现有的标头。这是默认值。 **replace**：设置标头，删除任何现有的标头。 **Never**：不设置标头，而是保留任何现有的标头。 **if-none**：如果没有设置标头，则设置它。	**ROUTER_SET_FORWARDED_HEADERS**

1. 如果允许列表中的 IP 地址和 CIDR 范围超过 61，它们将被写入到一个独立的文件中，haproxy.config 会引用这个文件。此文件存储在 var/lib/haproxy/router/whitelists 文件夹中。

注意

为确保地址被写入允许列表，请检查 Ingress Controller 配置文件中是否列出了 CIDR 范围的完整列表。etcd 对象大小限制了路由注解的大小。因此，它实际上是为您可以在允许列表中包含的 IP 地址和 CIDR 范围的最大数量创建一个阈值。



注意

环境变量不能编辑。

路由器超时变量

TimeUnits 由一个数字及一个时间单位表示：us *(microseconds), ms（毫秒，默认）、s（秒）、m（分钟）、h *(小时)、d （天）。

正则表达式是： [1-9][0-9]*(us\|ms\|s\|m\|h\|d)。

变量	默认	Description
ROUTER_BACKEND_CHECK_INTERVAL	5000ms	后端上后续存活度检查之间的时长。
ROUTER_CLIENT_FIN_TIMEOUT	1s	控制连接到路由的客户端的 TCP FIN 超时周期。如果发送到关闭连接的 FIN 在给定时间内没有回答，HAProxy 会关闭连接。如果设置为较低值，并且在路由器上使用较少的资源，则这不会产生任何损害。
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	客户端必须确认或发送数据的时长。
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	最长连接时间。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	控制路由器到支持路由的 pod 的 TCP FIN 超时。
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	服务器必须确认或发送数据的时长。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP 或 WebSocket 连接保持打开的时长。每当 HAProxy 重新加载时，这个超时期限都会重置。

变量	默认	Description
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	**300s**	设置等待出现新 HTTP 请求的最长时间。如果设置得太低，可能会导致浏览器和应用程序无法期望较小的 **keepalive** 值。 某些有效的超时值可以是某些变量的总和，而不是特定的预期超时。例如：**ROUTER_SLOWLORIS_HTTP_KEEPALIVE** 调整 **timeout http-keep-alive**。默认情况下，它设置为 **300s**，但 HAProxy 也会在 **tcp-request inspect-delay** 上等待，它被设置为 **5s**。在这种情况下，整个超时时间将是 **300s** 加 **5s**。
ROUTER_SLOWLORIS_TIMEOUT	**10s**	HTTP 请求传输可以花费的时间长度。
RELOAD_INTERVAL	**5s**	允许路由器至少执行重新加载和接受新更改的频率。
ROUTER_METRICS_HAPROXY_TIMEOUT	**5s**	收集 HAProxy 指标的超时时间。

设置自定义超时的路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
...
```

(1) **使用 HAProxy 支持的时间单位（us, ms, s, m, h, d）指定新的超时时间。如果没有提供时间单位，ms 会被默认使用。**

注意

如果为 passthrough 路由设置的服务器端的超时值太低，则会导致 WebSocket 连接在那个路由上经常出现超时的情况。

只允许一个特定 IP 地址的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

允许多个 IP 地址的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

允许 IP 地址 CIDR 网络的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

允许 IP 地址和 IP 地址 CIDR 网络的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

指定重写对象的路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / (1)
...
```

(1) 将 / 设为后端请求的重写路径。

在路由上设置 haproxy.router.openshift.io/rewrite-target 注解，指定 Ingress Controller 在将请求转发到后端应用程序之前，应该使用此路由在 HTTP 请求中重写路径。与 spec.path 中指定的路径匹配的请求路径部分将替换为注解中指定的重写对象。

下表提供了在 spec.path、请求路径和重写对象的各种组合中重写行为的路径示例。

表 28.4. rewrite-target 示例：

Route.spec.path	请求路径	重写目标	转发请求路径
/foo	/foo	/	/
/foo	/foo/	/	/
/foo	/foo/bar	/	/bar
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/
/foo	/foo/bar	/baz	/baz/bar

Route.spec.path	请求路径	重写目标	转发请求路径
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	不适用（请求路径不匹配路由路径）
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

haproxy.router.openshift.io/rewrite-target 中的某些特殊字符需要特殊处理，因为它们必须正确转义。请参阅下表以了解这些字符的处理方式。

表 28.5. 特殊字符处理：

对于字符	使用字符	注
#	\#	避免使用 #，因为它会终止重写表达式
%	% 或 %%	避免奇数序列，如 %%%
'	\'	避免 '，因为它被忽略

所有其他有效的 URL 字符可以在不转义的情况下使用。

28.1.11. 配置路由准入策略

管理员和应用程序开发人员可在多个命名空间中运行具有相同域名的应用程序。这是针对多个团队开发的、在同一个主机名上公开的微服务的机构。

警告

只有在命名空间间有信任的集群才会启用跨命名空间之间的声明，否则恶意用户可能会接管主机名。因此，默认的准入策略不允许在命名空间间声明主机名。

先决条件

- **必须具有集群管理员权限。**

流程

- **使用以下命令编辑 ingresscontroller 资源变量的.spec.** routeAdmission **字段：**

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
{"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --
type=merge
```

Ingress 控制器配置参数

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

提示

您还可以应用以下 YAML 来配置路由准入策略：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

28.1.12. 通过 Ingress 对象创建路由

一些生态系统组件与 Ingress 资源集成，但与路由资源不集成。要涵盖此问题单，OpenShift Container Platform 会在创建 Ingress 对象时自动创建受管路由对象。当相应 Ingress 对象被删除时，这些路由对象会被删除。

流程

1. 在 OpenShift Container Platform 控制台中或通过 oc create 命令来定义 Ingress 对象：

   Ingress 的 YAML 定义

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt" 1
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert 2
spec:
  rules:
  - host: www.example.com 3
    http:
      paths:
      - backend:
          service:
            name: frontend
            port:
              number: 443
```

```
      path: /
      pathType: Prefix
  tls:
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate
```

1 route.openshift.io/termination 注解可用于配置 Route 的 spec.tls.termination 字段，因为 Ingress 没有此字段。可接受的值为 edge、passthrough 和 reencrypt。所有其他值都会被静默忽略。当注解值未设置时，edge 是默认路由。模板文件中必须定义 TLS 证书详细信息，才能实现默认的边缘路由。

3 在使用 Ingress 对象时，您必须指定一个显式主机名，这与使用路由时不同。您可以使用 <host_name>.<cluster_ingress_domain> 语法（如 apps.openshiftdemos.com）以利用 *.<cluster_ingress_domain> 通配符 DNS 记录，为集群提供证书。否则，您必须确保有一个用于所选主机名的 DNS 记录。

a. 如果您在 route.openshift.io/termination 注解中指定 passthrough 值，在 spec 中将 path 设置为 ''，将 pathType 设置为 ImplementationSpecific：

```
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: ''
        pathType: ImplementationSpecific
        backend:
          service:
            name: frontend
            port:
              number: 443
```

```
$ oc apply -f ingress.yaml
```

2 route.openshift.io/destination-ca-certificate-secret 可用于 Ingress 对象来定义带有自定义目的地证书(CA)的路由。该注解引用一个 kubernetes secret，secret-ca-cert 将插入到生成的路由中。

a. 要从 ingress 对象使用目标 CA 指定路由对象，您必须在 secret 的 data.tls.crt specifier 中创建一个带有 PEM 编码格式的证书的 kubernetes.io/tls 或 Opaque 类型 secret。

2. 列出您的路由：

```
$ oc get routes
```

结果包括一个自动生成的路由，其名称以 frontend- 开头：

```
NAME            HOST/PORT       PATH    SERVICES    PORT    TERMINATION
WILDCARD
frontend-gnztq  www.example.com         frontend    443     reencrypt/Redirect  None
```

如果您检查这个路由，它会类似于：

自动生成的路由的 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
  ownerReferences:
  - apiVersion: networking.k8s.io/v1
    controller: true
    kind: Ingress
    name: frontend
    uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      [...]
      -----END RSA PRIVATE KEY-----
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
  to:
    kind: Service
    name: frontend
```

28.1.13. 通过 Ingress 对象使用默认证书创建路由

如果您在没有指定 TLS 配置的情况下创建 Ingress 对象，OpenShift Container Platform 会生成一个不安全的路由。要创建使用默认入口证书生成安全边缘终止路由的 Ingress 对象，您可以指定一个空的 TLS 配置，如下所示：

先决条件

- 您有一个要公开的服务。
- 您可以访问 OpenShift CLI(oc)。

流程

1. 为 Ingress 对象创建 YAML 文件。在本例中，该文件名为 **example-ingress.yaml** ：

   Ingress 对象的 YAML 定义

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:
  rules:
    ...
  tls:
  - {} 1
```

1 使用此精确的语法指定 TLS，而不指定自定义证书。

2. 运行以下命令来创建 Ingress 对象：

```
$ oc create -f example-ingress.yaml
```

验证

- 运行以下命令，验证 OpenShift Container Platform 是否为 Ingress 对象创建了预期的路由：

```
$ oc get routes -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd 1
    ...
  spec:
  ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
  ...
```

1 路由的名称包括 Ingress 对象的名称，后跟一个随机的后缀。

2 要使用默认证书，路由不应指定 spec.certificate。

3 路由应指定 edge 终止策略。

28.1.14. 在 Ingress 注解中使用目标 CA 证书创建路由

在 Ingress 对象上可以使用 route.openshift.io/destination-ca-certificate-secret 注解来定义带有自定义目标 CA 证书的路由。

先决条件

- 您可以在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须在 PEM 编码文件中有单独的目标 CA 证书。
- 您必须具有要公开的服务。

流程

1. 将 route.openshift.io/destination-ca-certificate-secret 添加到 Ingress 注解中：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert (1)
...
```

(1) 该注解引用 kubernetes secret。

2. 此注解中引用的机密将插入到生成的路由中。

   输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

28.1.15. 为双栈网络配置 OpenShift Container Platform Ingress Controller

如果您的 OpenShift Container Platform 集群是为 IPv4 和 IPv6 双栈网络配置的，则 OpenShift Container Platform 路由可从外部访问集群。

Ingress Controller 会自动提供具有 IPv4 和 IPv6 端点的服务，但您可以为单堆栈或双栈服务配置 Ingress Controller。

先决条件

- 您在裸机上部署了 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI（oc）。

流程

1. 要使 Ingress Controller 为工作负载提供通过 IPv4/IPv6 的流量，您可以通过设置 ipFamilies 和 ipFamilyPolicy 字段来创建服务 YAML 文件，或通过设置 ipFamilies 和 ipFamilyPolicy 字段来修改现有服务 YAML 文件。例如：

   服务 YAML 文件示例

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: yyyy-mm-ddT00:00:00Z
  labels:
    name: <service_name>
    manager: kubectl-create
    operation: Update
    time: yyyy-mm-ddT00:00:00Z
  name: <service_name>
  namespace: <namespace_name>
  resourceVersion: "<resource_version_number>"
  selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
  uid: <uid_number>
spec:
  clusterIP: 172.30.0.0/16
  clusterIPs: 1
  - 172.30.0.0/16
  - <second_IP_address>
  ipFamilies: 2
  - IPv4
  - IPv6
  ipFamilyPolicy: RequireDualStack 3
  ports:
  - port: 8080
    protocol: TCP
    targetport: 8080
  selector:
    name: <namespace_name>
  sessionAffinity: None
  type: ClusterIP
status:
  loadbalancer: {}
```

   1 在双栈实例中，提供了两个不同的 clusterIP。

   2 对于单堆栈实例，输入 IPv4 或 IPv6。对于双栈实例，请输入 IPv4 和 IPv6。

   3 对于单堆栈实例，请输入 SingleStack。对于双栈实例，请输入 RequireDualStack。

   这些资源生成对应的端点。Ingress Controller 现在监视endpointslices。

2. 要查看端点，请输入以下命令：

```
$ oc get endpoints
```

3. 要查看endpointslices，输入以下命令：

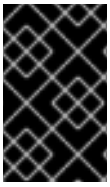
```
$ oc get endpointslices
```

其他资源

- 使用 appsDomain 选项指定备选集群域

28.2. 安全路由

安全路由提供以下几种 TLS 终止功能来为客户端提供证书。以下小节介绍了如何使用自定义证书创建重新加密、边缘和透传路由。



重要

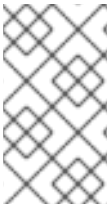
如果您在 Microsoft Azure 中创建通过公共端点的路由，则资源名称会受到限制。您不能创建使用某些词语的资源。如需 Azure 限制词语的列表，请参阅 Azure 文档中的解决预留资源名称错误。

28.2.1. 使用自定义证书创建重新加密路由

您可以通过 oc create route 命令，使用重新加密 TLS 终止和自定义证书配置安全路由。

前提条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须在 PEM 编码文件中有单独的目标 CA 证书。
- 您必须具有要公开的服务。



注意

不支持密码保护的密钥文件。要从密钥文件中删除密码，使用以下命令：

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

流程

此流程使用自定义证书和重新加密 TLS 终止创建 Route 资源。以下步骤假定证书/密钥对位于当前工作目录下的 tls.crt 和 tls.key 文件中。您还必须指定一个目标 CA 证书，使 Ingress Controller 信任服务的证书。您也可以根据需要指定 CA 证书来补全证书链。替换 tls.crt、tls.key、cacert.crt 和（可选）ca.crt 的实际路径名称。替换您要为 frontend 公开的 Service 资源的名称。使用适当的主机名替换 www.example.com。

- 使用重新加密 TLS 终止和自定义证书，创建安全 Route 资源：

```
$ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-
cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
```

如果您检查生成的 Route 资源，它应该类似于如下：

安全路由 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

如需了解更多选项，请参阅 oc create route reencrypt --help。

28.2.2. 使用自定义证书创建边缘路由

您可以通过 oc create route 命令，使用边缘 TLS 终止和自定义证书配置安全路由。使用边缘路由时，Ingress Controller 在将流量转发到目标 pod 之前终止 TLS 加密。该路由指定了 Ingress Controller 用于路由的 TLS 证书和密钥。

前提条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须具有要公开的服务。

注意

不支持密码保护的密钥文件。要从密钥文件中删除密码，使用以下命令：

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

流程

此流程使用自定义证书和边缘 TLS 终止创建 Route 资源。以下步骤假定证书/密钥对位于当前工作目录下的 tls.crt 和 tls.key 文件中。您也可以根据需要指定 CA 证书来补全证书链。替换 tls.crt、tls.key 和（可选）ca.crt 的实际路径名称。替换您要为 frontend 公开的服务名称。使用适当的主机名替换 www.example.com。

- 使用边缘 TLS 终止和自定义证书，创建安全 Route 资源。

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

如果您检查生成的 Route 资源，它应该类似于如下：

安全路由 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

如需了解更多选项，请参阅 oc create route edge --help。

28.2.3. 创建 passthrough 路由

您可以使用 oc create route 命令使用 passthrough 终止配置安全路由。如果 passthrough 终止，加密的流量会直接发送到目的地，而路由器不会提供 TLS 终止。因此，路由不需要密钥或证书。

前提条件

- 您必须具有要公开的服务。

流程

- 创建 Route 资源：

```
$ oc create route passthrough route-passthrough-secured --service=frontend --
port=8080
```

如果您检查生成的 Route 资源，它应该类似于如下：

使用 Passthrough 终止的安全路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough (2)
    insecureEdgeTerminationPolicy: None (3)
  to:
    kind: Service
    name: frontend
```

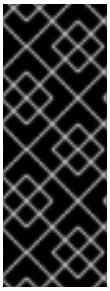
(1) 对象的名称，长度限于 63 个字符。

(2) termination 字段设置为 passthrough。这是唯一需要 tls 的字段。

(3) 可选的 insecureEdgeTerminationPolicy。禁用后唯一有效的值是 None、Redirect 或为空。

目标 pod 负责为端点上的流量提供证书。目前，这是唯一支持需要客户端证书的方法，也称双向验证。

28.2.4. 使用外部受管证书创建路由



重要

在 TLS secret 中使用外部证书保护路由只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以使用路由 API 的 .spec.tls.externalCertificate 字段来使用第三方证书管理解决方案配置 OpenShift Container Platform 路由。您可以通过 secret 引用外部管理的 TLS 证书，无需手动证书管理。使用外部受管证书可减少确保证书更新平稳推出的错误，使 OpenShift 路由器能够及时提供更新的证书。

注意

此功能适用于边缘路由和重新加密路由。

先决条件

- 您必须启用 RouteExternalCertificate 功能门。
- 您必须在 routes/custom-host 上具有 create 和 update 权限。
- 您必须有一个包含 PEM 编码格式的有效证书/密钥对的 secret，类型为 kubernetes.io/tls，其中包括 tls.key 和 tls.crt 键。
- 您必须将引用的 secret 放在与您要保护的路由相同的命名空间中。

流程

1. 运行以下命令，在与 secret 相同的命名空间中创建一个 角色，以允许路由器服务帐户读取访问权限：

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-
name=<secret-name> \ 1
--namespace=<current-namespace> 2
```

1 指定 secret 的实际名称。

2 指定 secret 和路由所在的命名空间。

2. 运行以下命令，在与 secret 相同的命名空间中创建 rolebinding，并将 router 服务帐户绑定到新创建的角色：

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace> 1
```

1 指定 secret 和路由所在的命名空间。

3. 创建一个定义 路由的 YAML 文件，并使用以下示例指定包含证书的 secret。

安全路由的 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: myedge
  namespace: test
spec:
  host: myedge-test.apps.example.com
  tls:
    externalCertificate:
      name: <secret-name> 1
    termination: edge
    [...]
[...]
```

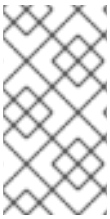
1 指定 secret 的实际名称。

4. 运行以下命令来创建一个路由资源：

```
$ oc apply -f <route.yaml> 1
```

1 指定生成的 YAML 文件名。

如果 secret 存在并具有证书/密钥对，如果满足所有先决条件，路由器将提供生成的证书。



注意

如果没有提供 .spec.tls.externalCertificate，路由器将使用默认生成的证书。

使用 .spec.tls.externalCertificate 字段时，您无法提供 .spec.tls.certificate 字段或 .spec.tls.key 字段。

其他资源

- 有关使用外部管理证书对路由进行故障排除，请检查 OpenShift Container Platform 路由器 pod 日志中的错误，请参阅 调查 pod 问题。

第 29 章 配置集群入口流量

29.1. 集群入口流量配置概述

OpenShift Container Platform 提供了以下从集群外部与集群中运行的服务进行通信的方法。

建议采用以下方法，它们按顺序或首选程度排列：

- **如果您有 HTTP/HTTPS，请使用 Ingress Controller。**
- **如果您有 HTTPS 之外的 TLS 加密协议。比如对于使用 SNI 标头的 TLS，请使用 Ingress Controller。**
- **否则，请使用负载均衡器、外部 IP 或 NodePort。**

方法	用途
使用 Ingress Controller	允许访问 HTTP/HTTPS 流量和 HTTPS 以外的 TLS 加密协议（例如，使用 SNI 标头的 TLS）。
使用负载均衡器服务自动分配外部 IP	允许流量通过从池分配的 IP 地址传到非标准端口。大多数云平台都提供了一种使用负载平衡器 IP 地址启动服务的方法。
关于 MetalLB 和 MetalLB Operator	允许从机器网络中的一个池到特定 IP 地址的流量。对于裸机安装或类似于裸机的平台，MetalLB 提供了使用负载平衡器 IP 地址启动服务的途径。
手动将外部 IP 分配给服务	允许流量通过特定的 IP 地址传到非标准端口。
配置 **NodePort**	在集群中的所有节点上公开某一服务。

29.1.1. 比较：对外部 IP 地址的容错访问

对于提供对外部 IP 地址访问权限的通信方法，另一个考虑因素是对 IP 地址的容错访问。以下功能提供对外部 IP 地址的容错访问。

IP 故障切换

IP 故障切换管理一组节点的虚拟 IP 地址池。它通过 Keepalived 和虚拟路由器冗余协议 (VRRP) 实施。IP 故障转移仅仅是第 2 层机制，它依赖于多播。对于某些网络，多播可能有缺点。

MetalLB

MetalLB 具有 2 层模式，但它不使用多播。第 2 层模式有一个缺点，它会通过一个节点传输外部 IP 地址的所有流量。

手动分配外部 IP 地址

您可以使用 IP 地址块配置集群，用于为服务分配外部 IP 地址。默认情况下禁用此功能。此功能非常灵活，但给集群或网络管理员带来了最大的负担。集群已准备好接收目标为外部 IP 的流量，但每个客户必须决定如何将流量路由到节点。

29.2. 为服务配置 EXTERNALIP

作为集群管理员，您可以指定可向集群中服务发送流量的集群外部 IP 地址块。

这个功能通常最适用于在裸机硬件上安装的集群。

29.2.1. 先决条件

- 您的网络基础架构必须将外部 IP 地址的流量路由到集群。

29.2.2. 关于 ExternalIP

对于非云环境，OpenShift Container Platform 支持通过 ExternalIP 工具将外部 IP 地址分配给 **Service** 对象的 **spec.externalIPs[]** 字段。通过设置此字段，OpenShift Container Platform 为服务分配额外的虚拟 IP 地址。IP 地址可以在为集群定义的服务网络之外。配置了 ExternalIP 功能的服务与具有 **type=NodePort** 的服务类似，允许您将流量定向到本地节点以进行负载均衡。

您必须配置网络基础架构，以确保您定义的外部 IP 地址块路由到集群。因此，从节点的网络接口中不会配置 IP 地址。要处理流量，您必须使用静态地址解析协议 (ARP) 条目等方法配置路由和访问外部 IP。

OpenShift Container Platform 通过添加以下功能来扩展 Kubernetes 中的 ExternalIP 功能：

- 通过可配置策略对用户使用外部 IP 地址的限制
- 根据请求自动将外部 IP 地址分配给服务

警告

默认情况下禁用，使用 ExternalIP 功能可能会造成安全隐患，因为集群内到一个外部 IP 地址的流量会定向到那个服务。这可让集群用户拦截用于外部资源的敏感流量。

重要

这个功能只在非云部署中被支持。对于云部署，使用负载均衡器服务自动部署云负载均衡器，以服务端点为目标。

您可以使用以下方法分配外部 IP 地址：

自动分配一个外部 IP

当创建了一个带有 **spec.type=LoadBalancer** 设置的 **Service** 对象时，OpenShift Container Platform 会从 **autoAssignCIDRs** CIDR 块中自动为 **spec.externalIPs[]** 分配一个 IP 地址。在本例中，OpenShift Container Platform 实现了负载均衡器服务类型的非云版本，并为服务分配 IP 地址。默认情况下，自动分配被禁用，且必须由集群管理员配置，如以下部分所述。

手动分配外部 IP

OpenShift Container Platform 在创建 **Service** 对象时使用分配给 **spec.externalIPs[]** 数组的 IP 地址。您不能指定已经被其他服务使用的 IP 地址。

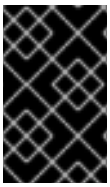
29.2.2.1. 配置 ExternalIP

在 OpenShift Container Platform 中使用外部 IP 地址取决于名为 **cluster** 的 **Network.config.openshift.io** CR 中的以下字段：

- spec.externalIP.autoAssignCIDRs 定义了一个负载均衡器在为服务选择外部 IP 地址时使用的 IP 地址块。OpenShift Container Platform 只支持单个 IP 地址块进行自动分配。当手工为服务分配 ExternalIPs 时，这比管理有限共享 IP 地址的端口空间更简单。如果启用了自动分配，则会为带有 spec.type=LoadBalancer 的 Service 对象分配一个外部 IP 地址。

- 在手动指定 IP 地址时，spec.externalIP.policy 定义了允许的 IP 地址块。OpenShift Container Platform 不会将策略规则应用到 spec.externalIP.autoAssignCIDRs 定义的 IP 地址块。

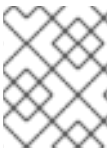
如果路由正确，来自配置的外部 IP 地址块的外部流量可以通过服务公开的任何 TCP 或 UDP 端口访问服务端点。



重要

作为集群管理员，您必须在 OpenShiftSDN 和 OVN-Kubernetes 网络类型中配置到 externalIPs 的路由。您还必须确保分配的 IP 地址块在集群中的一个或多个节点上终止。如需更多信息，请参阅 Kubernetes 外部 IP。

OpenShift Container Platform 支持自动和手动分配 IP 地址，并且保证每个地址都被分配到最多一个服务。这样可保证，无论由其他服务公开的端口是什么，每个服务都可以公开选择的端口。



注意

要使用 OpenShift Container Platform 中由 autoAssignCIDRs 定义 的 IP 地址块，您必须为主机网络配置必要的 IP 地址分配和路由。

以下 YAML 描述了配置了外部 IP 地址的服务：

带有 spec.externalIPs[] 设置的示例 Service 对象

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

29.2.2.2. 对外部 IP 地址分配的限制

作为集群管理员，您可以指定允许和拒绝的 IP 地址块。

限制只针对没有 cluster-admin 权限的用户。集群管理员始终可以将服务 spec.externalIPs[] 字段设置为任何 IP 地址。

您可以使用一个通过指定 spec.ExternalIP.policy 字段来定义的一个policy 对象来配置 IP 地址策略。策略对象有以下内容：

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

在配置策略限制时，会应用以下规则：

- 如果设置了 policy={}，那么创建带有 spec.ExternalIPs[] 设置的 Service 对象将失败。这是 OpenShift Container Platform 的默认设置。这与设置 policy=null 的行为相同。
- 如果设置了policy，并且设置了 policy.allowedCIDRs[] 或 policy.rejectedCIDRs[]，则应用以下规则：
  - 如果同时设置了 allowedCIDRs[] 和 rejectedCIDRs[]，则 allowedCIDRs[] 的设置高于 rejectedCIDRs[]。
  - 如果设置了 allowedCIDRs[]，只有在允许指定的 IP 地址时，创建带有 spec.ExternalIPs[] 的 Service 对象才能成功。
  - 如果设置了 rejectedCIDRs[]，只有在指定的 IP 地址未被拒绝时，创建带有 spec.ExternalIPs[] 的 Service 对象才能成功。

29.2.2.3. 策略对象示例

下面的例子演示了几个不同的策略配置。

- 在以下示例中，策略会防止 OpenShift Container Platform 使用指定的外部 IP 地址创建任何服务：

  拒绝为 Service 对象 spec.externalIPs[] 指定的任何值的策略示例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy: {}
    ...
  ```

- 在以下示例中，设置了 allowedCIDRs 和 rejectedCIDRs 字段。

  包括允许和拒绝 CIDR 块的策略示例

  ```
  ```

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
      - 172.16.66.10/23
      rejectedCIDRs:
      - 172.16.66.10/24
  ...
```

- 在以下示例中，policy 被设置为 null。如果设为 null，则通过输入 oc get network.config.openshift.io -o yaml 来检查配置对象时，policy 项不会出现在输出中。

  允许为 Service 对象 spec.externalIPs[]指定的任何值的示例策略

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...
```

29.2.3. ExternalIP 地址块配置

ExternalIP 地址块的配置由名为 cluster 的网络自定义资源（CR）定义。Network CR 是 config.openshift.io API 组的一部分。

重要

在集群安装过程中，Cluster Version Operator（CVO）会自动创建一个名为 cluster 的网络 CR。不支持创建此类型的任何其他 CR 对象。

以下 YAML 描述了 ExternalIP 配置：

network.config.openshift.io CR 名为 cluster

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
      ...
```

1 定义 CIDR 格式的 IP 地址块，可用于自动将外部 IP 地址分配给服务。只允许一个 IP 地址范围。

2 定义手动为服务分配 IP 地址的限制。如果没有定义限制，则不允许在 Service 对象中指定 spec.externalIP 字段。默认情况下，不会定义任何限制。

以下 YAML 描述了 policy 小节的字段：

network.config.openshift.io policy 小节

```
policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2
```

1 CIDR 格式允许的 IP 地址范围列表。

2 CIDR 格式拒绝的 IP 地址范围列表。

外部 IP 配置示例
以下示例中显示了外部 IP 地址池的一些可能配置：

- 以下 YAML 描述了启用自动分配外部 IP 地址的配置：

  带有 spec.externalIP.autoAssignCIDRs 的配置示例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
    - 192.168.132.254/29
```

- 以下 YAML 为允许的和被拒绝的 CIDR 范围配置策略规则：

  带有 spec.externalIP.policy 的示例配置

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    policy:
      allowedCIDRs:
      - 192.168.132.0/29
      - 192.168.132.8/29
      rejectedCIDRs:
      - 192.168.132.7/32
```

29.2.4. 为集群配置外部 IP 地址块

作为集群管理员，可以配置以下 ExternalIP 设置：

- OpenShift Container Platform 用来自动填充 Service 对象的 spec.clusterIP 字段的 ExternalIP 地址块。
- 用于限制可手动分配给 Service 对象的 spec.clusterIP 数组的 IP 地址的策略对象。

先决条件

- 安装 OpenShift CLI（oc）。
- 使用具有 cluster-admin 角色的用户访问集群。

流程

1. 可选：要显示当前的外部 IP 配置，请输入以下命令：

```
$ oc describe networks.config cluster
```

2. 要编辑配置，请输入以下命令：

```
$ oc edit networks.config cluster
```

3. 修改 ExternalIP 配置，如下例所示：

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
```

1 指定 externalIP 小节的配置。

4. 要确认更新的 ExternalIP 配置，请输入以下命令：

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

29.2.5. 后续步骤

- 为服务外部 IP 配置 ingress 集群流量

29.3. 使用 INGRESS CONTROLLER 配置集群入口流量

OpenShift Container Platform 提供了从集群外部与集群中运行的服务进行通信的方法。此方法使用了 Ingress Controller。

29.3.1. 使用 Ingress Controller 和路由

Ingress Operator 管理 Ingress Controller 和通配符 DNS。

使用 Ingress Controller 是允许从外部访问 OpenShift Container Platform 集群的最常用方法。

Ingress Controller 配置为接受外部请求并根据配置的路由进行代理。这仅限于 HTTP、使用 SNI 的 HTTPS 以及使用 SNI 的 TLS，对于通过使用 SNI 的 TLS 工作的 Web 应用程序和服务而言已经足够。

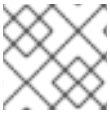
与管理员合作将 Ingress Controller 配置为接受外部请求并根据配置的路由进行代理。

管理员可以创建通配符 DNS 条目，再设置 Ingress Controller。然后，您可以处理边缘 Ingress Controller，无需与管理员联系。

默认情况下，集群中的每个 Ingress Controller 都可以接受集群中任何项目中创建的所有路由。

Ingress Controller：

- 默认有两个副本；即，它应该在两个 worker 节点上运行。
- 可以纵向扩张，以在更多节点上具有更多副本。



注意

这部分中的流程需要由集群管理员执行先决条件。

29.3.2. 先决条件

在开始以下流程前，管理员必须：

- 设置集群联网环境的外部端口，使请求能够到达集群。
- 确定至少有一个用户具有集群管理员角色。要将此角色添加到用户，请运行以下命令：

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- 有一个 OpenShift Container Platform 集群，其至少有一个 master 和至少一个节点，并且集群外有一个对集群具有网络访问权限的系统。此流程假设外部系统与集群位于同一个子网。不同子网上外部系统所需要的额外联网不在本主题的讨论范围内。

29.3.3. 创建项目和服务

如果您要公开的项目和服务尚不存在，请首先创建项目，再创建服务。

如果项目和服务都已存在，跳到公开服务以创建路由这一步。

先决条件

- 按照 oc CLI 并以一个集群管理员身份登陆。

流程

1. 运行 **oc new-project** 命令为您的服务创建一个新项目：

```
$ oc new-project myproject
```

2. 使用 **oc new-app** 命令来创建服务：

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. 要验证该服务是否已创建，请运行以下命令：

```
$ oc get svc -n myproject
```

输出示例

```
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex  ClusterIP  172.30.197.157  <none>        8080/TCP   70s
```

默认情况下，新服务没有外部 IP 地址。

29.3.4. 通过创建路由公开服务

您可以使用 oc expose 命令，将服务公开为路由。

流程

公开服务：

1. 登录 OpenShift Container Platform。
2. 登录您想公开的服务所在的项目：

```
$ oc project myproject
```

3. 运行 oc expose service 命令以公开路由：

```
$ oc expose service nodejs-ex
```

输出示例

```
route.route.openshift.io/nodejs-ex exposed
```

4. 要验证该服务是否已公开，您可以使用 cURL 等工具来确保该服务可从集群外部访问。
   a. 使用 oc get route 命令查找路由的主机名：

```
$ oc get route
```

输出示例

```
NAME       HOST/PORT                       PATH   SERVICES   PORT       TERMINATION   WILDCARD
nodejs-ex  nodejs-ex-myproject.example.com        nodejs-ex  8080-tcp                 None
```

   b. 使用 cURL 检查主机是否响应 GET 请求：

```
$ curl --head nodejs-ex-myproject.example.com
```

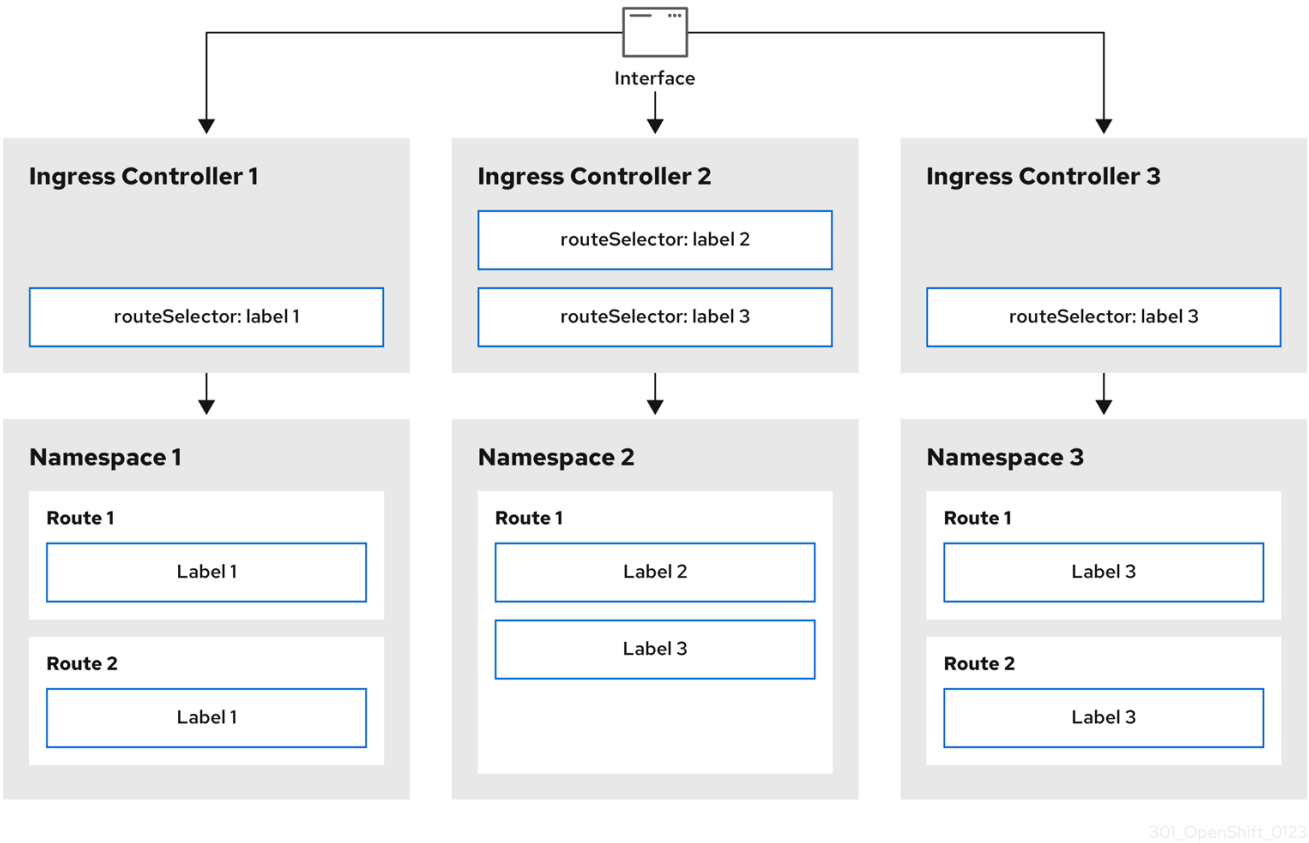
输出示例

```
HTTP/1.1 200 OK
...
```

29.3.5. 通过路由标签（label）配置 Ingress Controller 分片

使用路由标签进行 Ingress Controller 分片，意味着 Ingress Controller 提供由路由选择器选择的任意命名空间中的所有路由。

图 29.1. 使用路由标签进行 Ingress 分片



在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

流程

1. 编辑 router-internal.yaml 文件：

```
# cat router-internal.yaml
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
```

```
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
        type: sharded
```

1 指定 Ingress Controller 使用的域。此域必须与默认 Ingress Controller 域不同。

2. 应用 Ingress Controller **router-internal.yaml** 文件：

```
# oc apply -f router-internal.yaml
```

Ingress Controller 选择具有 **type: sharded** 标签的任意命名空间中的路由。

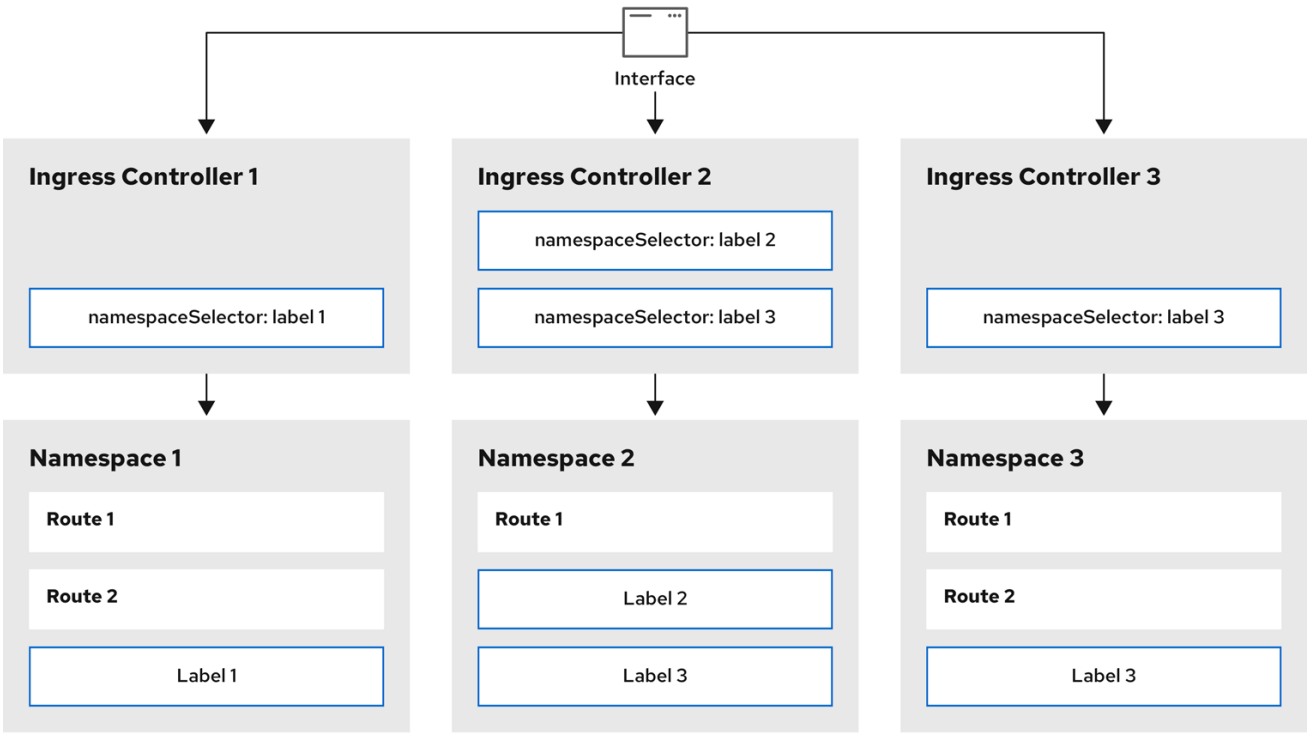
3. 使用 **router-internal.yaml** 中配置的域创建新路由：

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

29.3.6. 使用命名空间标签配置 Ingress Controller 分片

使用命名空间标签进行 Ingress Controller 分片，意味着 Ingress Controller 提供由命名空间选择器选择的任意命名空间中的所有路由。

图 29.2. 使用命名空间标签进行 Ingress 分片



在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

流程

1. 编辑 router-internal.yaml 文件：

```
# cat router-internal.yaml
```

输出示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

1 指定 Ingress Controller 使用的域。此域必须与默认 Ingress Controller 域不同。

2. 应用 Ingress Controller router-internal.yaml 文件：

```
# oc apply -f router-internal.yaml
```

Ingress Controller 选择由命名空间选择器选择的具有 type: sharded 标签的任意命名空间中的路由。

3. 使用 router-internal.yaml 中配置的域创建新路由：

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

29.3.7. 为 Ingress Controller 分片创建路由

通过使用路由，您可以通过 URL 托管应用程序。在这种情况下，主机名没有被设置，路由会使用子域。当您指定子域时，会自动使用公开路由的 Ingress Controller 域。对于由多个 Ingress Controller 公开路由的情况，路由由多个 URL 托管。

以下流程描述了如何为 Ingress Controller 分片创建路由，使用 hello-openshift 应用程序作为示例。

在一组 Ingress Controller 之间平衡传入的流量负载时，以及在将流量隔离到特定 Ingress Controller 时，Ingress Controller 分片会很有用处。例如，A 公司的流量使用一个 Ingress Controller，B 公司的流量则使用另外一个 Ingress Controller。

先决条件

- 已安装 OpenShift CLI（oc）。

- 您以项目管理员身份登录。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。
- 您已为分片配置了 Ingress Controller。

流程

1. 运行以下命令，创建一个名为 hello-openshift 的项目：

```
$ oc new-project hello-openshift
```

2. 运行以下命令，在项目中创建 pod：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-
openshift/hello-pod.json
```

3. 运行以下命令，创建名为 hello-openshift 的服务：

```
$ oc expose pod/hello-openshift
```

4. 创建名为 hello-openshift-route.yaml 的路由定义：

   为分片创建的路由的 YAML 定义：

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

   (1) 标签键及其对应标签值必须与 Ingress Controller 中指定的标签值匹配。在本例中，Ingress Controller 具有标签键和值 type: sharded。

   (2) 路由将使用 subdomain 字段的值公开。指定 subdomain 字段时，您必须保留主机名未设置。如果您同时指定了 host 和 subdomain 字段，则路由将使用 host 字段的值，并忽略 subdomain 字段。

5. 通过运行以下命令，使用 hello-openshift-route.yaml 创建到 hello-openshift 应用程序的路由：

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

验证

- 使用以下命令获取路由的状态：

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

生成的 Route 资源应类似以下示例：

输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-
sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1. Ingress Controller 或路由器的主机名用于公开路由。host 字段的值由 Ingress Controller 自动决定，并使用它的域。在本例中，Ingress Controller 的域为 <apps-sharded.basedomain.example.net>。
2. Ingress Controller 的主机名。
3. Ingress Controller 的名称。在本例中，Ingress Controller 的名称为 sharded。

29.3.8. 其他资源

Ingress Operator 管理通配符 DNS。如需更多信息，请参阅以下：

- OpenShift Container Platform 中的 Ingress Operator
- 在裸机上安装集群
- 在 vSphere 上安装集群
- 关于网络策略

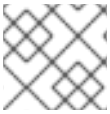
29.4. 使用负载均衡器配置集群入口流量

OpenShift Container Platform 提供了从集群外部与集群中运行的服务进行通信的方法。此方法使用了负载均衡器。

29.4.1. 使用负载均衡器使流量进入集群

如果不需要具体的外部 IP 地址，您可以配置负载均衡器服务，以便从外部访问 OpenShift Container Platform 集群。

负载均衡器服务分配唯一 IP。负载均衡器有单一边缘路由器 IP，它可以是虚拟 IP (VIP)，但仍然是一台用于初始负载均衡的计算机。



注意

如果配置了池，则会在基础架构一级进行，而不是由集群管理员完成。

注意

这部分中的流程需要由集群管理员执行先决条件。

29.4.2. 先决条件

在开始以下流程前，管理员必须：

- 设置集群联网环境的外部端口，使请求能够到达集群。
- 确定至少有一个用户具有集群管理员角色。要将此角色添加到用户，请运行以下命令：

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- 有一个 OpenShift Container Platform 集群，其至少有一个 master 和至少一个节点，并且集群外有一个对集群具有网络访问权限的系统。此流程假设外部系统与集群位于同一个子网。不同子网上外部系统所需要的额外联网不在本主题的讨论范围内。

29.4.3. 创建项目和服务

如果您要公开的项目和服务尚不存在，请首先创建项目，再创建服务。

如果项目和服务都已存在，跳到公开服务以创建路由这一步。

先决条件

- 按照 oc CLI 并以一个集群管理员身份登陆。

流程

1. 运行 **oc new-project** 命令为您的服务创建一个新项目：

```
$ oc new-project myproject
```

2. 使用 **oc new-app** 命令来创建服务：

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. 要验证该服务是否已创建，请运行以下命令：

```
$ oc get svc -n myproject
```

输出示例

```
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex   ClusterIP   172.30.197.157   <none>        8080/TCP   70s
```

默认情况下，新服务没有外部 IP 地址。

29.4.4. 通过创建路由公开服务

您可以使用 oc expose 命令，将服务公开为路由。

流程

公开服务：

1. 登录 OpenShift Container Platform。
2. 登录您想公开的服务所在的项目：

```
$ oc project myproject
```

3. 运行 oc expose service 命令以公开路由：

```
$ oc expose service nodejs-ex
```

输出示例

```
route.route.openshift.io/nodejs-ex exposed
```

4. 要验证该服务是否已公开，您可以使用 cURL 等工具来确保该服务可从集群外部访问。
   a. 使用 oc get route 命令查找路由的主机名：

```
$ oc get route
```

输出示例

```
NAME      HOST/PORT                   PATH   SERVICES    PORT      TERMINATION   WILDCARD
nodejs-ex   nodejs-ex-myproject.example.com          nodejs-ex   8080-tcp                None
```

   b. 使用 cURL 检查主机是否响应 GET 请求：

```
$ curl --head nodejs-ex-myproject.example.com
```

输出示例

```
HTTP/1.1 200 OK
...
```

29.4.5. 创建负载均衡器服务

使用以下流程来创建负载均衡器服务。

先决条件

- 确保您要公开的项目和服务已经存在。
- 您的云供应商支持负载均衡器。

流程

创建负载均衡器服务：

1. 登录 OpenShift Container Platform。
2. 加载您要公开的服务所在的项目。

```
$ oc project project1
```

3. 在 control plane 节点上打开文本文件并粘贴以下文本，根据需要编辑该文件：

负载均衡器配置文件示例

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 1
spec:
  ports:
  - name: db
    port: 3306 2
  loadBalancerIP:
  loadBalancerSourceRanges: 3
  - 10.0.0.0/8
  - 192.168.0.0/16
  type: LoadBalancer 4
  selector:
    name: mysql 5
```

1. 为负载均衡器服务输入一个描述性名称。
2. 输入您要公开的服务所侦听的同一个端口。
3. 输入特定 IP 地址列表来限制通过负载均衡器的流量。如果 cloud-provider 不支持这个功能，则此字段将被忽略。
4. 输入 Loadbalancer 作为类型。
5. 输入服务的名称。

注意

要将通过负载均衡器的流量限制到特定的 IP 地址，建议使用 Ingress Controller 字段 spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges。不要设置 loadBalancerSourceRanges 字段。

4. **保存并退出文件。**

5. **运行以下命令来创建服务：**

```
$ oc create -f <file-name>
```

例如：

```
$ oc create -f mysql-lb.yaml
```

6. **执行以下命令以查看新服务：**

```
$ oc get svc
```

输出示例

```
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP                         PORT(S)
AGE
egress-2  LoadBalancer   172.30.22.226  ad42f5d8b303045-487804948.example.com
3306:30357/TCP   15m
```

如果启用了云供应商，该服务会自动分配到一个外部 IP 地址。

7. **在 master 上，使用 cURL 等工具来确保您可以通过公共 IP 地址访问该服务：**

```
$ curl <public-ip>:<port>
```

例如：

```
$ curl 172.29.121.74:3306
```

此部分中的示例使用 MySQL 服务，这需要客户端应用程序。如果您得到一串字符并看到 Got packets out of order 消息，则您已连接到该服务：

如果您有 MySQL 客户端，请使用标准 CLI 命令登录：

```
$ mysql -h 172.30.131.89 -u admin -p
```

输出示例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

29.5. 在 AWS 上配置集群入口流量

OpenShift Container Platform 提供了从集群外部与集群中运行的服务进行通信的方法。此方法使用 AWS 上的负载均衡器，特别是 Network Load Balancer(NLB)或 Classic Load Balancer(CLB)。两种负载均衡器都可以将客户端的 IP 地址转发到节点，但 CLB 需要支持代理协议（OpenShift Container Platform 会自动启用）。

将 Ingress Controller 配置为使用 NLB 的方法有两种：

1. 通过强制替换当前使用 CLB 的 Ingress Controller。这会删除 **IngressController** 对象，并在新的 DNS 记录传播并置备 NLB 时发生停机。
2. 通过编辑使用 CLB 的现有 Ingress Controller 以使用 NLB。这会更改负载均衡器而无需删除并重新创建 **IngressController** 对象。

两种方法都可用于从 NLB 切换到 CLB。

您可以在新的或现有 AWS 集群上配置这些负载均衡器。

29.5.1. 在 AWS 中配置 Classic Load Balancer 超时

OpenShift Container Platform 提供了为特定路由或 Ingress Controller 设置自定义超时时间的方法。另外，AWS Classic Load Balancer(CLB)都有自己的超时时间，默认的超时时间为 60 秒。

如果 CLB 的超时时间小于路由超时或 Ingress Controller 超时，负载均衡器可以预先终止连接。您可以通过增加路由和 CLB 的超时周期来防止此问题。

29.5.1.1. 配置路由超时

如果您的服务需要低超时（满足服务级别可用性 (SLA) 目的）或高超时（具有慢速后端的情况），您可以为现有路由配置默认超时。

前提条件

- 您需要在运行的集群中部署了 Ingress Controller。

流程

1. 使用 **oc annotate** 命令，为路由添加超时：

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1 支持的时间单位是微秒 (us)、毫秒 (ms)、秒钟 (s)、分钟 (m)、小时 (h)、或天 (d)。

以下示例在名为 **myroute** 的路由上设置两秒的超时：

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

29.5.1.2. 配置 Classic Load Balancer 超时

您可以为 Classic Load Balancer(CLB)配置默认超时来扩展闲置连接。

先决条件

- 您必须在正在运行的集群中部署了 Ingress Controller。

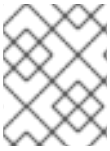
流程

1. 运行以下命令，为默认的 ingresscontroller 设置 AWS 连接闲置超时：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"type":"LoadBalancerService", "loadBalancer": \
    {"scope":"External", "providerParameters":{"type":"AWS", "aws": \
    {"type":"Classic", "classicLoadBalancer": \
    {"connectionIdleTimeout":"5m"}}}}}}'
```

2. 可选：运行以下命令来恢复超时的默认值：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"providerParameters":{"aws":{"classicLoadBalancer": \
    {"connectionIdleTimeout":null}}}}}}'
```



注意

在更改连接超时值时，您必须指定 scope 字段，除非已经设置了当前范围。设置 scope 字段时，如果恢复默认的超时值，则不需要再次这样做。

29.5.2. 使用网络负载平衡器在 AWS 上配置集群入口流量

OpenShift Container Platform 提供了从集群外部与集群中运行的服务进行通信的方法。一个这样的方法是使用 Network Load Balancer(NLB)。您可以在新的或现有 AWS 集群上配置 NLB。

29.5.2.1. 将 Ingress Controller 从使用 Classic Load Balancer 切换到网络负载均衡器

您可以将使用 Classic Load Balancer (CLB) 的 Ingress Controller 切换到使用 AWS 上的网络负载平衡器 (NLB) 的 Ingress Controller。

在这些负载均衡器间切换不会删除 IngressController 对象。

警告

此过程可能会导致以下问题：

- 由于新的 DNS 记录传播、新的负载均衡器置备和其他因素而可能需要几分钟的中断。应用此步骤后，Ingress Controller 负载均衡器的 IP 地址和规范名称可能会改变。
- 由于服务注解的变化，会泄漏负载均衡器资源。

流程

1. **修改您要使用 NLB 切换到的现有 Ingress Controller。这个示例假定您的默认 Ingress Controller 具有外部范围，且没有其他自定义：**

   ingresscontroller.yaml 文件示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

注意

如果您没有为 spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type 字段指定值，Ingress Controller 会使用在安装过程中设置的集群Ingress 配置中的 spec.loadBalancer.platform.aws.type 值。

提示

如果您的 Ingress Controller 有其他要更新的自定义（如更改域），请考虑强制替换 Ingress Controller 定义文件。

2. **运行以下命令，将更改应用到 Ingress Controller YAML 文件：**

```
$ oc apply -f ingresscontroller.yaml
```

当 Ingress Controller 更新时，可能会有几分钟的停机。

29.5.2.2. 将 Ingress Controller 从使用 Network Load Balancer 切换到使用 Classic Load Balancer

在 AWS 中，您可以将使用 Network Load Balancer (NLB) 的 Ingress Controller 切换到使用 Classic Load Balancer (CLB) 的 Ingress Controller。

在这些负载均衡器间切换不会删除 IngressController 对象。

警告

此流程会导致预期的中断会因为新的 DNS 记录传播、新的负载均衡器置备和其他因素而可能需要几分钟。应用此步骤后，Ingress Controller 负载均衡器的 IP 地址和规范名称可能会改变。

流程

1. **修改您要切换为使用 CLB 的现有 Ingress Controller。这个示例假定您的默认 Ingress Controller 具有外部范围，且没有其他自定义：**

   ingresscontroller.yaml 文件示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: Classic
    type: LoadBalancerService
```

注意

如果您没有为 spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type 字段指定值，Ingress Controller 会使用在安装过程中设置的集群Ingress 配置中的 spec.loadBalancer.platform.aws.type 值。

提示

如果您的 Ingress Controller 有其他要更新的自定义（如更改域），请考虑强制替换 Ingress Controller 定义文件。

2. **运行以下命令，将更改应用到 Ingress Controller YAML 文件：**

```
$ oc apply -f ingresscontroller.yaml
```

当 Ingress Controller 更新时，可能会有几分钟的停机。

29.5.2.3. 将 Ingress Controller Classic Load Balancer 替换为网络负载均衡器

您可以将使用 Classic 负载平衡器(CLB)的 Ingress Controller 替换为 AWS 上使用网络负载平衡器(NLB)的 Ingress Controller。

警告

此过程可能会导致以下问题：

- 由于新的 DNS 记录传播、新的负载均衡器置备和其他因素而可能需要几分钟的中断。应用此步骤后，Ingress Controller 负载均衡器的 IP 地址和规范名称可能会改变。
- 由于服务注解的变化，会泄漏负载均衡器资源。

流程

1. 创建一个新的默认 Ingress Controller 文件。以下示例假定您的默认 Ingress Controller 具有外部范围，且没有其他自定义：

   ingresscontroller.yml 文件示例

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     creationTimestamp: null
     name: default
     namespace: openshift-ingress-operator
   spec:
     endpointPublishingStrategy:
       loadBalancer:
         scope: External
         providerParameters:
           type: AWS
           aws:
             type: NLB
       type: LoadBalancerService
   ```

   如果您的默认 Ingress Controller 有其他自定义，请确定您相应地修改该文件。

   提示

   如果您的 Ingress Controller 没有其他自定义，且您只更新负载均衡器类型，请考虑 "Switch the Ingress Controller from using an Classic Load Balancer to a Network Load Balancer" 中详述的步骤。

2. 强制替换 Ingress Controller YAML 文件：

   ```
   $ oc replace --force --wait -f ingresscontroller.yml
   ```

   等待 Ingress Controller 已被替换。预计会有几分钟的停机时间。

29.5.2.4. 在现有 AWS 集群上配置 Ingress Controller 网络负载均衡器

您可以在当前集群中创建一个由 AWS Network Load Balancer（NLB）支持的 Ingress Controller。

先决条件

- 您必须已安装 AWS 集群。
- 基础架构资源的 PlatformStatus 需要是 AWS。
  - 要验证 PlatformStatus 是否为 AWS，请运行：

```
$ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'
AWS
```

流程

在现有集群中，创建一个由 AWS NLB 支持的 Ingress Controller。

1. 创建 Ingress Controller 清单：

```
$ cat ingresscontroller-aws-nlb.yaml
```

输出示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: $my_ingress_controller 1
  namespace: openshift-ingress-operator
spec:
  domain: $my_unique_ingress_domain 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External 3
      providerParameters:
        type: AWS
        aws:
          type: NLB
```

1. 将 $my_ingress_controller 替换为 Ingress Controller 的唯一名称。
2. 将 $my_unique_ingress_domain 替换为集群中所有 Ingress Controller 的唯一域名。此变量需要是 DNS 名的子域 <clustername>.<domain>。
3. 您可以将 External 替换为 Internal，以使用内部 NLB。

2. 在集群中创建资源：

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```

重要

在新 AWS 集群上配置 Ingress Controller NLB 之前，您必须完成 创建安装配置文件 的步骤。

29.5.2.5. 在新 AWS 集群上配置 Ingress Controller 网络负载平衡

您可在新集群中创建一个由 AWS Network Load Balancer（NLB）支持的 Ingress Controller。

先决条件

- 创建 install-config.yaml 文件并完成对其所做的任何修改。

流程

在新集群中，创建一个由 AWS NLB 支持的 Ingress Controller。

1. 进入包含安装程序的目录并创建清单：

```
$ ./openshift-install create manifests --dir <installation_directory> 1
```

1 对于 <installation_directory>，请指定含有集群的 install-config.yaml 文件的目录的名称。

2. 在 <installation_directory>/manifests/ 目录中创建一个名为 cluster-ingress-default-ingresscontroller.yaml 的文件：

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml 1
```

1 对于 <installation_directory>，请指定包含集群的 manifests/ 目录的目录名称。

创建该文件后，几个网络配置文件位于 manifests/ 目录中，如下所示：

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

输出示例

```
cluster-ingress-default-ingresscontroller.yaml
```

3. 在编辑器中打开 cluster-ingress-default-ingresscontroller.yaml 文件，并输入描述您想要的 Operator 配置的自定义资源（CR）：

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
```

```
    scope: External
    providerParameters:
      type: AWS
      aws:
        type: NLB
  type: LoadBalancerService
```

4. 保存 cluster-ingress-default-ingresscontroller.yaml 文件并退出文本编辑器。

5. 可选：备份 manifests/cluster-ingress-default-ingresscontroller.yaml 文件。创建集群时，安装程序会删除 manifests/ 目录。

29.5.3. 其他资源

- 使用自定义网络在 AWS 上安装集群。
- 如需有关对 NLBs 的支持的更多信息，请参阅 AWS 上的网络负载平衡支持。
- 如需有关 CLBs 的代理协议支持的更多信息，请参阅为您的 Classic Load Balancer 配置代理协议支持

29.6. 为服务外部 IP 配置 INGRESS 集群流量

您可以将外部 IP 地址附加到服务，使其可用于集群外的流量。这通常只适用于在裸机硬件上安装的集群。必须正确配置外部网络基础架构，将流量路由到该服务。

29.6.1. 先决条件

- 您的集群被配置为启用了 ExternalIP。如需更多信息，请参阅为服务配置 ExternalIPs。

注意

对于 egress IP，不要使用相同的 ExternalIP。

29.6.2. 将 ExternalIP 附加到服务

您可以将 ExternalIP 附加到服务。如果您的集群被配置为自动分配 ExternalIP，您可能不需要手动将 ExternalIP 附加到该服务。

流程

1. 可选：要确认为 ExternalIP 配置了哪些 IP 地址范围，请输入以下命令：

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
```

如果设置了 autoAssignCIDRs，在没有指定 spec.externalIPs 字段的情况下，OpenShift Container Platform 会自动为新的 Service 对象分配一个 ExternalIP。

2. 为服务附加一个 ExternalIP。

   a. 如果要创建新服务，请指定 spec.externalIPs 字段，并提供包括一个或多个有效 IP 地址的数组。例如：

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externalIPs:
  - 192.174.120.10
```

b. 如果您要将 ExternalIP 附加到现有服务中，请输入以下命令。将 <name> 替换为服务名称。将 <ip_address> 替换为有效的 ExternalIP 地址。您可以提供多个以逗号分开的 IP 地址。

```
$ oc patch svc <name> -p \
  '{
    "spec": {
      "externalIPs": [ "<ip_address>" ]
    }
  }'
```

例如：

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

输出示例

```
"mysql-55-rhel7" patched
```

3. 要确认一个 ExternalIP 地址已附加到该服务，请输入以下命令。如果为新服务指定 ExternalIP，您必须首先创建该服务。

```
$ oc get svc
```

输出示例

```
NAME            CLUSTER-IP      EXTERNAL-IP     PORT(S)    AGE
mysql-55-rhel7    172.30.131.89   192.174.120.10  3306/TCP   13m
```

29.6.3. 其他资源

- 为服务配置 ExternalIP

29.7. 使用 NODEPORT 配置集群入口流量

OpenShift Container Platform 提供了从集群外部与集群中运行的服务进行通信的方法。此方法使用了 NodePort。

29.7.1. 使用 NodePort 使流量进入集群

使用 NodePort 类型的 Service 资源，在集群中所有节点的特定端口上公开服务。端口在 Service 资源的 .spec.ports[*].nodePort 字段中指定。

重要

使用节点端口需要额外的端口资源。

NodePort 在节点 IP 地址的静态端口上公开服务。默认情况下，NodePort 在 30000 到 32767 的范围内，这意味着，NodePort 不可能与服务的预期端口匹配。例如：端口 8080 可能会在节点的端口 31020 中公开。

管理员必须确保外部 IP 地址路由到节点。

NodePort 和外部 IP 地址互相独立，可以同时使用它们。

注意

这部分中的流程需要由集群管理员执行先决条件。

29.7.2. 先决条件

在开始以下流程前，管理员必须：

- 设置集群联网环境的外部端口，使请求能够到达集群。
- 确定至少有一个用户具有集群管理员角色。要将此角色添加到用户，请运行以下命令：

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- 有一个 OpenShift Container Platform 集群，其至少有一个 master 和至少一个节点，并且集群外有一个对集群具有网络访问权限的系统。此流程假设外部系统与集群位于同一个子网。不同子网上外部系统所需要的额外联网不在本主题的讨论范围内。

29.7.3. 创建项目和服务

如果您要公开的项目和服务尚不存在，请首先创建项目，再创建服务。

如果项目和服务都已存在，跳到公开服务以创建路由这一步。

先决条件

- 按照 oc CLI 并以一个集群管理员身份登陆。

流程

1. 运行 **oc new-project** 命令为您的服务创建一个新项目：

```
$ oc new-project myproject
```

2. 使用 **oc new-app** 命令来创建服务：

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. 要验证该服务是否已创建，请运行以下命令：

```
$ oc get svc -n myproject
```

输出示例

```
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
nodejs-ex   ClusterIP   172.30.197.157   <none>       8080/TCP   70s
```

默认情况下，新服务没有外部 IP 地址。

29.7.4. 通过创建路由公开服务

您可以使用 oc expose 命令，将服务公开为路由。

流程

公开服务：

1. 登录 OpenShift Container Platform。
2. 登录您想公开的服务所在的项目：

```
$ oc project myproject
```

3. 要为应用程序公开节点端口，请输入以下命令修改服务的自定义资源定义 (CRD)：

```
$ oc edit svc <service_name>
```

输出示例

```
spec:
  ports:
  - name: 8443-tcp
    nodePort: 30327 1
    port: 8443
    protocol: TCP
    targetPort: 8443
  sessionAffinity: None
  type: NodePort 2
```

1 可选：指定应用程序的节点端口范围。默认情况下，OpenShift Container Platform 在 30000-32767 范围内选择一个可用端口。

2 定义服务类型。

4. 可选：要使用公开的节点端口确认该服务可用，请输入以下命令：

```
$ oc get svc -n myproject
```

输出示例

```
NAME              TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
nodejs-ex         ClusterIP   172.30.217.127   <none>       3306/TCP         9m44s
nodejs-ex-ingress   NodePort    172.30.107.72    <none>       3306:31345/TCP   39s
```

5. 可选：要删除由 oc new-app 命令自动创建的服务，请输入以下命令：

```
$ oc delete svc nodejs-ex
```

验证

- 要检查服务节点端口是否已使用 30000-32767 范围内的端口更新，请输入以下命令：

```
$ oc get svc
```

在以下示例输出中，更新的端口为 30327 ：

输出示例

```
NAME    TYPE      CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
httpd   NodePort  172.xx.xx.xx   <none>        8443:30327/TCP  109s
```

29.7.5. 其他资源

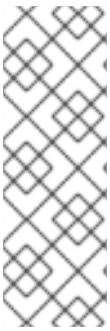
- 配置节点端口服务范围

29.8. 使用负载均衡器允许的源范围配置集群入口流量

您可以为 IngressController 指定 IP 地址范围列表。这在 endpointPublishingStrategy 为 LoadBalancerService 时，限制访问负载均衡器服务。

29.8.1. 配置负载均衡器允许的源范围

您可以启用并配置 spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges 字段。通过配置负载均衡器允许的源范围，您可以将 Ingress Controller 的负载均衡器的访问限制为指定的 IP 地址范围列表。Ingress Operator 协调负载均衡器服务，并根据 AllowedSourceRanges 设置 spec.loadBalancerSourceRanges 字段。



注意

如果您已经在 OpenShift Container Platform 的早期版本中设置了 spec.loadBalancerSourceRanges 字段或负载均衡器服务 anotation service.beta.kubernetes.io/load-balancer-source-ranges，Ingress Controller 会在升级后开始报告 Progressing=True。要解决这个问题，设置覆盖 spec.loadBalancerSourceRanges 字段的 AllowedSourceRanges，并清除 service.beta.kubernetes.io/load-balancer-source-ranges 注解。Ingress Controller 开始报告 Progressing=False。

先决条件

- 您需要在正在运行的集群中部署了 Ingress Controller。

流程

- 运行以下命令，为 Ingress Controller 设置允许的源范围 API：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
   --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
   {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}' 1
```

1 示例值 0.0.0.0/0 指定允许的源范围。

29.8.2. 迁移到允许的源范围

如果您已经设置了注解 service.beta.kubernetes.io/load-balancer-source-ranges，您可以迁移到允许负载均衡器的源范围。当您设置 AllowedSourceRanges 时，Ingress Controller 根据 AllowedSourceRanges 值设置 spec.loadBalancerSourceRanges 字段，并取消设置 service.beta.kubernetes.io/load-balancer-source-ranges 注解。

注意

如果您已经在 OpenShift Container Platform 的早期版本中设置了 spec.loadBalancerSourceRanges 字段或负载均衡器服务 anotation service.beta.kubernetes.io/load-balancer-source-ranges，Ingress Controller 会在升级后开始报告 Progressing=True。要解决这个问题，设置覆盖 spec.loadBalancerSourceRanges 字段的 AllowedSourceRanges，并清除 service.beta.kubernetes.io/load-balancer-source-ranges 注解。Ingress Controller 再次开始报告 Progressing=False。

先决条件

- 您已设置了 service.beta.kubernetes.io/load-balancer-source-ranges 注解。

流程

1. 确保设置了 service.beta.kubernetes.io/load-balancer-source-ranges ：

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

输出示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/load-balancer-source-ranges: 192.168.0.1/32
```

2. 确保 spec.loadBalancerSourceRanges 字段已取消设置：

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

输出示例

```
...
spec:
  loadBalancerSourceRanges:
  - 0.0.0.0/0
...
```

3. 将集群更新至 OpenShift Container Platform 4.16。
4. 运行以下命令，为 ingresscontroller 设置允许的源范围 API：

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}' (1)
```

(1) 示例值 0.0.0.0/0 指定允许的源范围。

29.8.3. 其他资源

- OpenShift 更新简介

第 30 章 KUBERNETES NMSTATE

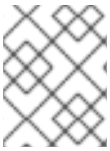
30.1. 关于 KUBERNETES NMSTATE OPERATOR

Kubernetes NMState Operator 提供了一个 Kubernetes API，用于使用 NMState 在 OpenShift Container Platform 集群的节点上执行状态驱动的网络配置。Kubernetes NMState Operator 为用户提供了在集群节点上配置各种网络接口类型、DNS 和路由的功能。另外，集群节点中的守护进程会定期向 API 服务器报告每个节点的网络接口状态。

重要

红帽在使用裸机、IBM Power®、IBM Z®、IBM® LinuxONE、VMware vSphere 和 OpenStack 安装的生产环境中支持 Kubernetes NMState Operator。

在 OpenShift Container Platform 中使用 NMState 之前，必须安装 Kubernetes NMState Operator。



注意

Kubernetes NMState Operator 更新二级 NIC 的网络配置。它无法更新主 NIC 或 **br-ex** 网桥的网络配置。

OpenShift Container Platform 使用 nmstate 来报告并配置节点网络的状态。这样便可通过将单个配置清单应用到集群来修改网络策略配置，例如在所有节点上创建 Linux 网桥。

节点网络由以下对象监控和更新：

NodeNetworkState

报告该节点上的网络状态。

NodeNetworkConfigurationPolicy

描述节点上请求的网络配置。您可以通过将 **NodeNetworkConfigurationPolicy**清单应用到集群来更新节点网络配置，包括添加和删除网络接口 。

NodeNetworkConfigurationEnactment

报告每个节点上采用的网络策略。

30.1.1. 安装 Kubernetes NMState Operator

您可以使用 web 控制台或 CLI 安装 Kubernetes NMState Operator。

30.1.1.1. 使用 Web 控制台安装 Kubernetes NMState Operator

您可以使用 web 控制台安装 Kubernetes NMState Operator。安装后，Operator 可将 NMState State Controller 部署为在所有集群节点中的守护进程集。

先决条件

- 您以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 选择 Operators → OperatorHub。

2. 在 All Items 下面的搜索字段中, 输入 **nmstate** 并点 Enter 来搜索 Kubernetes NMState Operator。
3. 点 Kubernetes NMState Operator 搜索结果。
4. 点 Install 打开 Install Operator 窗口。
5. 点 Install 安装 Operator。
6. Operator 安装完成后，点 View Operator。
7. 在 Provided APIs 下，点 Create Instance 打开对话框以创建 **kubernetes-nmstate** 实例。
8. 在对话框的 Name 字段中，确保实例的名称是 **nmstate.**

注意

名称限制是一个已知问题。该实例是整个集群的单个实例。

9. 接受默认设置并点 Create 创建实例。

概述

完成后，Operator 将 NMState State Controller 部署为在所有集群节点中的守护进程集。

30.1.1.2. 使用 CLI 安装 Kubernetes NMState Operator

您可以使用 OpenShift CLI(**oc**) 安装 Kubernetes NMState Operator。安装后，Operator 可将 NMState State Controller 部署为在所有集群节点中的守护进程集。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 创建 **nmstate** Operator 命名空间：

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-nmstate
    name: openshift-nmstate
  name: openshift-nmstate
spec:
  finalizers:
  - kubernetes
EOF
```

2. 创建 **OperatorGroup** ：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: NMState.v1.nmstate.io
  name: openshift-nmstate
  namespace: openshift-nmstate
spec:
  targetNamespaces:
  - openshift-nmstate
EOF
```

3. 订阅 **nmstate** Operator:

```
$ cat << EOF| oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/kubernetes-nmstate-operator.openshift-nmstate: ""
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kubernetes-nmstate-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. 确认 **nmstate** Operator 部署的 **ClusterServiceVersion** (CSV) 状态等于 **Succeeded** ：

```
$ oc get clusterserviceversion -n openshift-nmstate \
 -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

输出示例

```
Name                                         Phase
kubernetes-nmstate-operator.4.16.0-202210210157   Succeeded
```

5. 创建 **nmstate** Operator 实例：

```
$ cat << EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
EOF
```

6. 验证 NMState Operator 的 pod 是否正在运行：

```
$ oc get pod -n openshift-nmstate
```

输出示例

```
Name                                  Ready   Status  Restarts  Age
pod/nmstate-cert-manager-5b47d8dddf-5wnb5   1/1   Running  0       77s
pod/nmstate-console-plugin-d6b76c6b9-4dcwm  1/1   Running  0       77s
pod/nmstate-handler-6v7rm                   1/1   Running  0       77s
pod/nmstate-handler-bjcxw                   1/1   Running  0       77s
pod/nmstate-handler-fv6m2                   1/1   Running  0       77s
pod/nmstate-handler-kb8j6                   1/1   Running  0       77s
pod/nmstate-handler-wn55p                   1/1   Running  0       77s
pod/nmstate-operator-f6bb869b6-v5m92        1/1   Running  0       4m51s
pod/nmstate-webhook-66d6bbd84b-6n674        1/1   Running  0       77s
pod/nmstate-webhook-66d6bbd84b-vlzrd        1/1   Running  0       77s
```

30.2. 观察和更新节点网络状态和配置

30.2.1. 使用 CLI 查看节点的网络状态

节点网络状态是集群中所有节点的网络配置。一个 NodeNetworkState 对象存在于集群中的每个节点上。此对象定期更新，并捕获该节点的网络状态。

流程

1. 列出集群中的所有 NodeNetworkState 对象：

```
$ oc get nns
```

2. 检查 NodeNetworkState 对象以查看该节点上的网络。为了清楚，这个示例中的输出已被重新编辑：

```
$ oc get nns node01 -o yaml
```

输出示例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  name: node01 (1)
status:
  currentState: (2)
    dns-resolver:
# ...
    interfaces:
# ...
    route-rules:
# ...
    routes:
# ...
  lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" (3)
```

(1) NodeNetworkState 对象的名称从节点获取。

2 currentState 包含节点的完整网络配置，包括 DNS、接口和路由。

3 最新成功更新的时间戳。只要节点可以被访问，这个时间戳就会定期更新，它可以用来指示报告的新旧程度。

30.2.2. 从 Web 控制台查看节点的网络状态

作为管理员，您可以使用 OpenShift Container Platform Web 控制台观察 NodeNetworkState 资源和网络接口，并访问网络详情。

流程

1. 进入到 Networking → NodeNetworkState。
   在 NodeNetworkState 页面中，您可以查看 NodeNetworkState 资源列表以及节点上创建的对应接口。您可以基于接口状态, 接口类型, 和 IP 进行过滤，或者使用基于条件名称或标签的搜索栏来缩小显示的 NodeNetworkState 资源范围。
2. 要访问 NodeNetworkState 资源的详细信息，请点 Name 列中列出的 NodeNetworkState 资源名称。
3. 要展开并查看 NodeNetworkState 资源的 Network Details 部分，点 > 图标。或者，您也可以点 Network interface 列下的每个接口类型来查看网络详情。

30.2.3. 从 Web 控制台管理策略

您可以通过将 NodeNetworkConfigurationPolicy 清单应用到集群来更新节点网络的配置，如为节点添加或删除接口。通过访问 Networking 菜单下的 NodeNetworkConfigurationPolicy 页面中创建的策略列表，从 web 控制台管理策略。通过此页面，您可以创建、更新、监控和删除策略。

30.2.3.1. 监控策略状态

您可以在 NodeNetworkConfigurationPolicy 页面中监控策略状态。本页以表格格式显示集群中创建的所有策略，其列如下：

Name
: 创建的策略的名称。

匹配的节点
: 应用策略的节点计数。这可能是基于节点选择器或集群中的所有节点的节点子集。

节点网络状态
: 匹配节点的 Enactment 状态。您可以点 enactment 状态并查看状态的详细信息。

要查找所需的策略，您可以使用 Filter 选项或搜索选项根据 enactment 状态过滤列表。

30.2.3.2. 创建策略

您可以使用 web 控制台中的表单或 YAML 创建策略。

流程

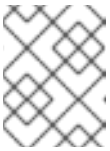
1. 进入到 Networking → NodeNetworkConfigurationPolicy。
2. 在 NodeNetworkConfigurationPolicy 页面中，点 Create，然后选择 From Form选项。

如果没有现有策略，您可以使用表单点 Create NodeNetworkConfigurationPolicy 来创建策略。

注意

要使用 YAML 创建策略，请点 Create，然后选择 With YAML 选项。以下步骤仅适用于使用表单创建策略。

3. 可选：选择 Apply this NodeNetworkConfigurationPolicy only to specific subsets of nodes using the node selector 复选框，以指定必须应用策略的节点。
4. 在 Policy name 字段中输入策略名称。
5. 可选：在 Description 字段中输入策略的描述。
6. 可选：在 Policy Interface (s)部分中，默认添加了一个桥接接口，带有可编辑的字段中的预设置值。通过执行以下步骤来编辑值：
   a. 在 Interface name 字段中输入接口名称。
   b. 从 Network 状态下拉菜单中选择网络状态。默认选择的值是 Up。
   c. 从类型下拉菜单中选择接口类型。可用的值包括 Bridge, Bonding, 和 Ethernet。默认选择的值是 Bridge。



注意

不支持使用表单添加 VLAN 接口。要添加 VLAN 接口，您必须使用 YAML 来创建策略。添加后，您无法使用表单编辑策略。

   d. 可选： 在 IP 配置部分中，选中 IPv4 复选框，为接口分配 IPv4 地址，并配置 IP 地址分配详情：
      i. 点IP 地址，将接口配置为使用静态 IP 地址；或点 DHCP 将接口配置为自动分配 IP 地址。
      ii. 如果您选择了 IP 地址 选项，在 IPV4 address 字段中输入 IPv4 地址，并在 Prefix length 字段中输入前缀长度。
      如果您选择了 DHCP 选项，请取消选择您要禁用的选项。可用选项包括 Auto-DNS、Auto-routes 和 Auto-gateway。所有选项都被默认选择。
   e. 可选：在 Port 字段中输入端口号。
   f. 可选： 选中启用 STP复选框以启用 STP。
   g. 可选： 要为策略添加一个接口，点 Add another interface to the policy。

h. 可选： 要从策略中删除接口，点接口旁的

n. 可选：要从策略中删除接口，点接口旁的


图标。

注意

或者，您可以点页面顶部的 Edit YAML 继续使用 YAML 编辑表单。

7. 点 Create 以完成策略创建。

30.2.3.3. 更新策略

30.2.3.3.1. 使用表单更新策略

流程

1. 进入到 Networking → NodeNetworkConfigurationPolicy。
2. 在 NodeNetworkConfigurationPolicy 页面中，点您要编辑的策略旁的  图标，然后点 Edit。
3. 编辑您要更新的字段。
4. 点击 Save。

注意

不支持使用表单添加 VLAN 接口。要添加 VLAN 接口，您必须使用 YAML 来创建策略。添加后，您无法使用表单编辑策略。

30.2.3.3.2. 使用 YAML 更新策略

流程

1. 进入到 Networking → NodeNetworkConfigurationPolicy。
2. 在 NodeNetworkConfigurationPolicy 页面中，点您要编辑的策略的 Name 列下的策略名称。
3. 点 YAML 选项卡，并编辑 YAML。
4. 点击 Save。

30.2.3.4. 删除策略

流程

1. 进入到 Networking → NodeNetworkConfigurationPolicy。
2. 在 NodeNetworkConfigurationPolicy 页面中，点您要删除的策略旁的  图标，然后点 Delete。
3. 在弹出窗口中，输入策略名称以确认删除，然后点 Delete。

30.2.4. 使用 CLI 管理策略

30.2.4.1. 在节点上创建接口

通过将一个 NodeNetworkConfigurationPolicy 清单应用到集群来在集群的节点上创建一个接口。清单详细列出了请求的接口配置。

默认情况下，清单会应用到集群中的所有节点。要将接口只添加到特定的节点，在节点选择器上添加 spec: nodeSelector 参数和适当的 <key>:<value> 。

您可以同时配置多个支持 nmstate 节点。该配置适用于并行节点的 50%。如果网络连接失败，此策略可防止整个集群不可用。要将策略配置并行应用到集群的特定部分，请使用 maxUnavailable 字段。

流程

1. 创建 NodeNetworkConfigurationPolicy 清单。以下示例在所有 worker 节点上配置了一个 Linux 桥接并配置 DNS 解析器：

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3 (4)
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port (5)
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
          auto-dns: false
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1
    dns-resolver: (6)
      config:
```

```
      search:
      - example.com
      - example.org
      server:
      - 8.8.8.8
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 本例使用 node-role.kubernetes.io/worker："" 节点选择器来选择集群中的所有 worker 节点。
4. 可选：指定策略配置可同时应用到的最大 nmstate 节点数。这个参数可以设置为百分比值（字符串），如 "10%"，也可以是绝对值（数字），如 3。
5. 可选：接口人类可读的描述。
6. 可选：指定 DNS 服务器的搜索和服务器设置。

2. 创建节点网络策略：

```
$ oc apply -f br1-eth1-policy.yaml 1
```

1. 节点网络配置策略清单的文件名。

其他资源

- 在相同策略中创建多个接口的示例
- 策略中不同 IP 管理方法示例

30.2.4.2. 确认节点上的节点网络策略更新

NodeNetworkConfigurationPolicy 清单描述了您为集群中的节点请求的网络配置。节点网络策略包括您请求的网络配置以及整个集群中的策略执行状态。

当您应用节点网络策略时，会为集群中的每个节点创建一个 NodeNetworkConfigurationEnactment 对象。节点网络配置是一个只读对象，代表在该节点上执行策略的状态。如果策略在节点上应用失败，则该节点会包括 traceback 用于故障排除。

流程

1. 要确认策略已应用到集群，请列出策略及其状态：

```
$ oc get nncp
```

2. 可选：如果策略配置成功的时间比预期的要长，您可以检查特定策略请求的状态和状态条件：

```
$ oc get nncp <policy> -o yaml
```

3. 可选：如果策略在所有节点上配置成功的时间比预期的要长，您可以列出集群中的 Enactments 的状态：

```
$ oc get nnce
```

4. 可选：要查看特定的 Enactment 的配置，包括对失败配置进行任何错误报告：

```
$ oc get nnce <node>.<policy> -o yaml
```

30.2.4.3. 从节点中删除接口

您可以通过编辑 NodeNetworkConfigurationPolicy 对象从集群中的一个或多个节点中删除接口，并将接口的状态设置为 absent。

从节点中删除接口不会自动将节点网络配置恢复到以前的状态。如果要恢复之前的状态，则需要在策略中定义节点网络配置。

如果删除了网桥或绑定接口，以前附加到该网桥或绑定接口的任何节点 NIC 都会处于 down 状态并变得不可访问。为了避免连接丢失，在相同策略中配置节点 NIC，使其具有 up 状态，以及使用 DHCP 或一个静态 IP 地址。

注意

删除添加接口的节点网络策略不会更改节点上的策略配置。虽然 NodeNetworkConfigurationPolicy 是集群中的一个对象，但它只代表请求的配置。同样，删除接口不会删除策略。

流程

1. 更新用来创建接口的 NodeNetworkConfigurationPolicy 清单。以下示例删除了 Linux 网桥，并使用 DHCP 配置 eth1 NIC 以避免断开连接：

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
    - name: br1
      type: linux-bridge
      state: absent 4
    - name: eth1 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 本例使用 node-role.kubernetes.io/worker：""节点选择器来选择集群中的所有 worker 节

4 将状态改为 absent 会删除接口。

5 要从网桥接口中取消附加的接口名称。

6 接口的类型。这个示例创建了以太网网络接口。

7 接口的请求状态。

8 可选：如果您不使用 dhcp，可以设置静态 IP，或让接口没有 IP 地址。

9 在这个示例中启用 ipv4 。

2. 更新节点上的策略并删除接口：

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

1 策略清单的文件名。

30.2.5. 不同接口的策略配置示例

以下示例显示了不同的 NodeNetworkConfigurationPolicy 清单配置。

为了获得最佳性能，请在应用策略时请考虑以下因素：

- 当您需要将策略应用到多个节点时，为每个目标节点创建一个 NodeNetworkConfigurationPolicy 清单。将策略限定到单个节点可减少 Kubernetes NMState Operator 应用策略的总时长。
  相反，如果单个策略包含多个节点的配置，Kubernetes NMState Operator 会按顺序将策略应用到每个节点，这会增加策略应用程序的整体长度。

- 所有相关的网络配置都应在单一策略中指定。
  当节点重启时，Kubernetes NMState Operator 无法控制应用策略的顺序。因此，Kubernetes NMState Operator 可能会按顺序应用相互独立的策略，从而导致网络对象降级。

30.2.5.1. 示例： Linux bridge interface 节点网络配置策略

通过将一个 NodeNetworkConfigurationPolicy 清单应用到集群来在集群的节点上创建一个 Linux 网桥接口。

以下 YAML 文件是 Linux 网桥界面的清单示例。如果运行 playbook，其中会包含必须替换为您自己的信息的样本值。

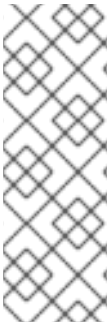
```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: br1 4
        description: Linux bridge with eth1 as a port 5
```

```
    type: linux-bridge 6
    state: up 7
    ipv4:
      dhcp: true 8
      enabled: true 9
    bridge:
      options:
        stp:
          enabled: false 10
      port:
        - name: eth1 11
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 这个示例使用 hostname 节点选择器。
4. 接口的名称。
5. 可选：接口人类可读的接口描述。
6. 接口的类型。这个示例会创建一个桥接。
7. 创建后接口的请求状态。
8. 可选：如果您不使用 dhcp，可以设置静态 IP，或让接口没有 IP 地址。
9. 在这个示例中启用 ipv4。
10. 在这个示例中禁用 stp。
11. 网桥附加到的节点 NIC。

30.2.5.2. 示例：VLAN 接口节点网络配置策略

通过将一个 NodeNetworkConfigurationPolicy 清单应用到集群来在集群的节点上创建一个 VLAN 接口。



注意

在单个 NodeNetworkConfigurationPolicy 清单中为节点的 VLAN 接口定义所有相关配置。例如，在同一 NodeNetworkConfigurationPolicy 清单中为节点定义 VLAN 接口和相关路由。

当节点重启时，Kubernetes NMState Operator 无法控制应用策略的顺序。因此，如果您将单独的策略用于相关的网络配置，Kubernetes NMState Operator 可能会按顺序应用这些策略，从而导致网络对象降级。

以下 YAML 文件是 VLAN 接口的清单示例。如果运行 playbook，其中会包含必须替换为您自己的信息的样本值。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
```

```
metadata:
  name: vlan-eth1-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
    - name: eth1.102 (4)
      description: VLAN using eth1 (5)
      type: vlan (6)
      state: up (7)
      vlan:
        base-iface: eth1 (8)
        id: 102 (9)
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 这个示例使用 hostname 节点选择器。
4. 接口的名称。当在裸机上部署时，只支持 <interface_name>.<vlan_number> VLAN 格式。
5. 可选：接口人类可读的接口描述。
6. 接口的类型。这个示例创建了一个 VLAN。
7. 创建后接口的请求状态。
8. 附加 VLAN 的节点 NIC。
9. VLAN 标签。

30.2.5.3. 示例：虚拟功能的节点网络配置策略（技术预览）

通过应用 NodeNetworkConfigurationPolicy 清单，更新现有集群中的单根 I/O 虚拟化 (SR-IOV) 网络功能 (VF) 的主机网络设置。

重要

为 SR-IOV 网络 VF 更新主机网络设置只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以将 NodeNetworkConfigurationPolicy 清单应用到现有集群，以完成以下任务：

- 为 VF 配置 QoS 或 MTU 主机网络设置，以优化性能。
- 为网络接口添加、删除或更新 VF。
- 管理 VF 绑定配置。

注意

要在也通过 SR-IOV Network Operator 管理的物理功能上使用 NMState 更新 SR-IOV VF 的主机网络设置，您必须将相关 SriovNetworkNodePolicy 资源中的 externallyManaged 参数设置为 true。如需更多信息，请参阅*附加资源部分*。

以下 YAML 文件是一个清单示例，它为 VF 定义 QoS 策略。此文件包含必须替换为您自己的信息的样本值。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: qos 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
      - name: ens1f0 4
        description: Change QOS on VF0 5
        type: ethernet 6
        state: up 7
        ethernet:
          sr-iov:
            total-vfs: 3 8
            vfs:
            - id: 0 9
              max-tx-rate: 200 10
```

1. **策略的名称。**
2. **可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。**
3. **本例适用于具有 worker 角色的所有节点。**
4. **物理功能(PF)网络接口的名称。**
5. **可选：接口人类可读的接口描述。**
6. **接口的类型。**
7. **配置后接口的请求状态。**
8. **VF 的总数。**
9. **标识 ID 为 0 的 VF。**
10. **为 VF 设置最大传输率（以 Mbps 为单位）。此示例值设置 200 Mbps 的速度。**

以下 YAML 文件是一个清单示例，它在 VF 上创建 VLAN 接口并将其添加到绑定的网络接口中。如果运行 playbook，其中会包含必须替换为您自己的信息的样本值。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
```

```
metadata:
  name: addvf (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens1f0v1 (4)
        type: ethernet
        state: up
      - name: ens1f0v1.477 (5)
        type: vlan
        state: up
        vlan:
          base-iface: ens1f0v1 (6)
          id: 477
      - name: bond0 (7)
        description: Add vf (8)
        type: bond (9)
        state: up (10)
        link-aggregation:
          mode: active-backup (11)
          options:
            primary: ens1f1v0.477 (12)
          port: (13)
            - ens1f1v0.477
            - ens1f0v0.477
            - ens1f0v1.477 (14)
```

(1) 策略的名称。

(2) 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。

(3) 本例适用于具有 worker 角色的所有节点。

(4) VF 网络接口的名称。

(5) VLAN 网络接口的名称。

(6) 附加 VLAN 接口的 VF 网络接口。

(7) 绑定网络接口的名称。

(8) 可选：接口人类可读的接口描述。

(9) 接口的类型。

(10) 配置后接口的请求状态。

(11) 绑定的绑定策略。

(12) 主附加绑定端口。

(13) 绑定网络接口的端口。

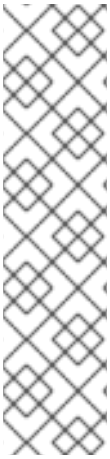
14 在本例中，此 VLAN 网络接口作为额外接口添加到绑定网络接口。

其他资源

- 配置 SR-IOV 网络设备

30.2.5.4. 示例：绑定接口节点网络配置策略

通过将一个 NodeNetworkConfigurationPolicy 清单应用到集群来在集群的节点上创建一个绑定接口。



注意

OpenShift Container Platform 只支持以下绑定模式：

- mode=1 active-backup
- mode=2 balance-xor
- mode=4 802.3ad
- mode=5 balance-tlb
- mode=6 balance-alb

以下 YAML 文件是绑定接口的清单示例。如果运行 playbook，其中会包含必须替换为您自己的信息的样本值。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond0 4
      description: Bond with ports eth1 and eth2 5
      type: bond 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
      link-aggregation:
        mode: active-backup 10
        options:
          miimon: '140' 11
        port: 12
        - eth1
        - eth2
      mtu: 1450 13
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 这个示例使用 hostname 节点选择器。
4. 接口的名称。
5. 可选：接口人类可读的接口描述。
6. 接口的类型。这个示例创建了一个绑定。
7. 创建后接口的请求状态。
8. 可选：如果您不使用 dhcp，可以设置静态 IP，或让接口没有 IP 地址。
9. 在这个示例中启用 ipv4 。
10. Bond 的驱动模式。这个示例使用 active 备份模式。
11. 可选：本例使用 miimon 检查每 140ms 的绑定链接。
12. 绑定中的下级节点 NIC。
13. 可选：绑定的最大传输单元（MTU）。如果没有指定，其默认值为 1500。

30.2.5.5. 示例：以太网接口节点网络配置策略

通过将 NodeNetworkConfigurationPolicy 清单应用到集群，在集群的节点上配置以太网接口。

以下 YAML 文件是一个以太接口的清单示例。它包含了示例值，需要使用自己的信息替换。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1 4
      description: Configuring eth1 on node01 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1. 策略的名称。
2. 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。
3. 这个示例使用 hostname 节点选择器。

4 接口的名称。

5 可选：接口人类可读的接口描述。

6 接口的类型。这个示例创建了以太网网络接口。

7 创建后接口的请求状态。

8 可选：如果您不使用 dhcp，可以设置静态 IP，或让接口没有 IP 地址。

9 在这个示例中启用 ipv4 。

30.2.5.6. 示例：同一节点网络配置策略中的多个接口

您可以在相同的节点网络配置策略中创建多个接口。这些接口可以相互引用，允许您使用单个策略清单来构建和部署网络配置。

以下示例 YAML 文件在两个 NIC 和 VLAN 之间创建一个名为 bond10 的绑定，名为 bond10.103，它连接到绑定。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond-vlan 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond10 4
      description: Bonding eth2 and eth3 5
      type: bond 6
      state: up 7
      link-aggregation:
        mode: balance-rr 8
        options:
          miimon: '140' 9
        port: 10
        - eth2
        - eth3
    - name: bond10.103 11
      description: vlan using bond10 12
      type: vlan 13
      state: up 14
      vlan:
        base-iface: bond10 15
        id: 103 16
      ipv4:
        dhcp: true 17
        enabled: true 18
```

1 策略的名称。

2 可选： 如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。

3 这个示例使用 hostname 节点选择器。

4 11 接口的名称。

5 12 可选：接口人类可读的接口描述。

6 13 接口的类型。

7 14 创建后接口的请求状态。

8 Bond 的驱动模式。

9 可选：本例使用 miimon 检查每 140ms 的绑定链接。

10 绑定中的下级节点 NIC。

15 附加 VLAN 的节点 NIC。

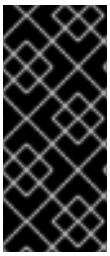
16 VLAN 标签。

17 可选：如果您不使用 dhcp，可以设置静态 IP，或者让接口没有 IP 地址。

18 在这个示例中启用 ipv4。

30.2.5.7. 示例：带有 VRF 实例网络配置策略的网络接口

通过应用 NodeNetworkConfigurationPolicy 自定义资源(CR)将虚拟路由和转发(VRF)实例与网络接口关联。



重要

将 VRF 实例与网络接口关联只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

通过将 VRF 实例与网络接口关联，您可以支持流量隔离、独立路由决策和网络资源的逻辑分离。

在裸机环境中，您可以使用 MetalLB 通过属于 VRF 实例的接口宣布负载均衡器服务。如需更多信息，请参阅*附加资源部分*。

以下 YAML 文件是一个将 VRF 实例与网络接口关联的示例。如果运行 playbook，其中会包含必须替换为您自己的信息的样本值。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy 1
spec:
  nodeSelector:
    vrf: "true" 2
  maxUnavailable: 3
```

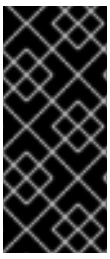
```
desiredState:
  interfaces:
    - name: ens4vrf 3
      type: vrf 4
      state: up
      vrf:
        port:
          - ens4 5
        route-table-id: 2 6
```

1. 策略的名称。
2. 这个示例将策略应用到带有 vrf:true 标签的所有节点。
3. 接口的名称。
4. 接口的类型。这个示例创建了一个 VRF 实例。
5. VRF 附加到的节点接口。
6. VRF 的路由表 ID 的名称。

其他资源

- 关于虚拟路由和转发
- 通过网络 VRF 公开服务

30.2.6. 捕获附加到网桥的 NIC 的静态 IP



重要

捕获 NIC 的静态 IP 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

30.2.6.1. 示例：Linux 网桥接口节点网络配置策略，用于从附加到网桥的 NIC 中继承静态 IP 地址

在集群的节点上创建一个 Linux 网桥接口，并通过将单个 NodeNetworkConfigurationPolicy 清单应用到集群来将 NIC 的静态 IP 配置传输到桥接。

以下 YAML 文件是 Linux 网桥界面的清单示例。它包含了示例值，需要使用自己的信息替换。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-copy-ipv4-policy 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: ""
```

```
  capture:
    eth1-nic: interfaces.name=="eth1" 3
    eth1-routes: routes.running.next-hop-interface=="eth1"
    br1-routes: capture.eth1-routes | routes.running.next-hop-interface := "br1"
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge 4
        state: up
        ipv4: "{{ capture.eth1-nic.interfaces.0.ipv4 }}" 5
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1 6
    routes:
      config: "{{ capture.br1-routes.routes.running }}"
```

1. 策略的名称。
2. 可选：如果没有包括 nodeSelector 参数，策略会应用到集群中的所有节点。本例使用 node-role.kubernetes.io/worker："" 节点选择器来选择集群中的所有 worker 节点。
3. 对网桥附加的节点 NIC 的引用。
4. 接口的类型。这个示例会创建一个桥接。
5. 网桥接口的 IP 地址。这个值与 spec.capture.eth1-nic 条目引用的 NIC 的 IP 地址匹配。
6. 网桥附加到的节点 NIC。

其他资源

- NMPolicy 项目 - 策略语法

30.2.7. 示例：IP 管理

以下配置片段示例演示了不同的 IP 管理方法。

这些示例使用 ethernet 接口类型来简化示例，同时显示 Policy 配置中相关的上下文。这些 IP 管理示例可与其他接口类型一起使用。

30.2.7.1. Static

以下片段在以太网接口中静态配置 IP 地址：

```
# ...
    interfaces:
    - name: eth1
      description: static IP on eth1
      type: ethernet
      state: up
```

```
    ipv4:
      dhcp: false
      address:
      - ip: 192.168.122.250 1
        prefix-length: 24
      enabled: true
# ...
```

1 使用接口的静态 IP 地址替换这个值。

30.2.7.2. 没有 IP 地址

以下片段确保接口没有 IP 地址：

```
# ...
  interfaces:
  - name: eth1
    description: No IP on eth1
    type: ethernet
    state: up
    ipv4:
      enabled: false
# ...
```

30.2.7.3. 动态主机配置

以下片段配置了一个以太网接口，它使用动态 IP 地址、网关地址和 DNS：

```
# ...
  interfaces:
  - name: eth1
    description: DHCP on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      enabled: true
# ...
```

以下片段配置了一个以太网接口，它使用动态 IP 地址，但不使用动态网关地址或 DNS：

```
# ...
  interfaces:
  - name: eth1
    description: DHCP without gateway or DNS on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
      auto-dns: false
      enabled: true
# ...
```

30.2.7.4. DNS

默认情况下，nmstate API 会在全局范围内存储 DNS 值，而不是将其存储在网络接口中。在某些情况下，您必须配置网络接口来存储 DNS 值。要为网络接口定义 DNS 配置，您必须首先在网络接口 YAML 配置文件中指定 dns-resolver 部分。

提示

设置一个 DNS 配置与修改 /etc/resolv.conf 文件相当。

重要

在配置 DNS 解析器时，您不能使用 br-ex 网桥（一个 OVNKubernetes 管理的 Open vSwitch 网桥）作为接口。

以下示例显示了全局存储 DNS 值的默认情况：

- 配置没有网络接口的静态 DNS。请注意，当更新主机节点上的 /etc/resolv.conf 文件时，您不需要在 NodeNetworkConfigurationPolicy (NNCP)清单中指定一个接口 (IPv4 或 IPv6)。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
 name: worker-0-dns-testing
spec:
  nodeSelector:
    kubernetes.io/hostname: <target_node>
  desiredState:
    dns-resolver:
      config:
        search:
        - example.com
        - example.org
        server:
        - 2001:db8:f::1
        - 192.0.2.251
# ...
```

以下示例显示，配置一个网络接口来存储 DNS 值：

- 为具有自动 IP 配置的网络接口配置静态 DNS。请注意，对于此配置，您必须将 auto-dns 参数设置为 false，以便 Kubernetes NMState Operator 可以为网络接口的存储自定义 DNS 设置。

```
dns-resolver:
  config:
    search:
    - example.com
    - example.org
    server:
    - 2001:db8:f::1
    - 192.0.2.251
interfaces:
  - name: eth1
    type: ethernet
    state: up
```

```
    ipv4:
      enabled: true
      dhcp: true
      auto-dns: false
    ipv6:
      enabled: true
      dhcp: true
      autoconf: true
      auto-dns: false
# ...
```

- 为带有静态 IP 配置的网络接口配置静态 DNS。请注意，对于此配置，您必须将 dhcp 参数设置为 false，并将 autoconf 参数设置为 false。

```
dns-resolver:
  config:
# ...
    server:
    - 2001:4860:4860::8844
    - 192.0.2.251
interfaces:
  - name: eth1
    type: ethernet
    state: up
    ipv4:
      enabled: true
      dhcp: false
      address:
      - ip: 192.0.2.251
        prefix-length: 24
    ipv6:
      enabled: true
      dhcp: false
      autoconf: false
      address:
      - ip: 2001:db8:1::1
        prefix-length: 64
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.1
    next-hop-interface: eth1
  - destination: ::/0
    next-hop-address: 2001:db8:1::3
    next-hop-interface: eth1
# ...
```

- 配置一个静态 DNS 名称服务器，附加到 DHCP 和 IPv6 Stateless Address AutoConfiguration (SLAAC) 服务器。

```
dns-resolver:
  config:
# ...
    server:
    - 192.0.2.251
```

```
interfaces:
  - name: eth1
    type: ethernet
    state: up
    ipv4:
      enabled: true
      dhcp: true
      auto-dns: true
    ipv6:
      enabled: true
      dhcp: true
      autoconf: true
      auto-dns: true
# ...
```

30.2.7.5. 静态路由

以下片段在接口 eth1 中配置静态路由和静态 IP。

```
dns-resolver:
  config:
# ...
interfaces:
  - name: eth1
    description: Static routing on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: false
      enabled: true
      address:
      - ip: 192.0.2.251 (1)
        prefix-length: 24
routes:
  config:
  - destination: 198.51.100.0/24
    metric: 150
    next-hop-address: 192.0.2.1 (2)
    next-hop-interface: eth1
    table-id: 254
# ...
```

(1) 以太网接口的静态 IP 地址。

(2) 节点流量的下一跳地址。这必须与为以太接口设定的 IP 地址位于同一个子网中。

30.3. 对节点网络配置进行故障排除

如果节点网络配置遇到问题，则策略会自动回滚，且报告失败。这包括如下问题：

- 配置没有在主机上应用。
- 主机丢失了到默认网关的连接。

- 断开了与 API 服务器的连接。

30.3.1. 对不正确的节点网络配置策略配置进行故障排除

您可以通过应用节点网络配置策略，对整个集群中的节点网络配置应用更改。如果应用了不正确的配置，您可以使用以下示例进行故障排除并修正失败的节点网络策略。

在本例中，一个 Linux 桥接策略应用到一个具有三个 control plane 节点和三个计算节点的示例集群。策略无法应用，因为它会引用了一个不正确的接口。要查找错误，调查可用的 NMState 资源。然后您可以使用正确配置来更新策略。

流程

1. 创建策略并将其应用到集群。以下示例在 ens01 接口上创建了一个简单桥接：

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

输出示例

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. 运行以下命令，验证策略的状态：

```
$ oc get nncp
```

输出显示策略失败：

输出示例

```
NAME                STATUS
ens01-bridge-testfail   FailedToConfigure
```

但是，仅有策略状态并不表示它在所有节点或某个节点子集中是否失败。

3. 列出节点网络配置以查看策略在任意节点上是否成功。如果策略只针对某个节点子集失败，这表示问题在于特定的节点配置。如果策略在所有节点上都失败，这表示问题在于策略。

```
$ oc get nnce
```

输出显示策略在所有节点上都失败：

输出示例

```
NAME                                    STATUS
control-plane-1.ens01-bridge-testfail   FailedToConfigure
control-plane-2.ens01-bridge-testfail   FailedToConfigure
control-plane-3.ens01-bridge-testfail   FailedToConfigure
compute-1.ens01-bridge-testfail         FailedToConfigure
compute-2.ens01-bridge-testfail         FailedToConfigure
compute-3.ens01-bridge-testfail         FailedToConfigure
```

4. 查看失败的原因之一并查看回溯信息。以下命令使用输出工具 jsonpath 来过滤输出结果：

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?
(@.type=="Failing")].message}'
```

这个命令会返回一个大的回溯信息，它被编辑为 brevity:

输出示例

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to
execute nmstatectl set --no-commit --timeout 480: 'exit status 1' ''
...
libnmstate.error.NmstateVerificationError:
desired
=======
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port:
  - name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
```

```
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=======
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

difference
==========
--- desired
+++ current
@@ -13,8 +13,7 @@
       hello-time: 2
       max-age: 20
       priority: 32768
-  port:
-  - name: ens01
+  port: []
 description: Linux bridge with the wrong port
 ipv4:
   address: []
  line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:
```

NmstateVerificationError 列出了 desired（期望的） 策略配置，策略在节点上的current（当前的） 配置，并高亮标识了不匹配参数间的difference（不同）。在本示例中，此 port 包含在 difference 部分，这表示策略中的端口配置问题。

5. 要确保正确配置了策略，请求 NodeNetworkState 来查看一个或多个节点的网络配置。以下命令返回 control-plane-1 节点的网络配置：

```
$ oc get nns control-plane-1 -o yaml
```

输出显示节点上的接口名称为 ens1，但失败的策略使用了 ens01:

输出示例

```
  - ipv4:
# ...
    name: ens1
    state: up
    type: ethernet
```

6. 通过编辑现有策略修正错误：

```
$ oc edit nncp ens01-bridge-testfail
```

```
# ...
        port:
         - name: ens1
```

保存策略以应用更正。

7. 检查策略的状态，以确保它被成功更新：

```
$ oc get nncp
```

输出示例

```
NAME                   STATUS
ens01-bridge-testfail   SuccessfullyConfigured
```

在集群中的所有节点上都成功配置了更新的策略。

第 31 章 配置集群范围代理

生产环境可能会拒绝直接访问互联网，而是提供 HTTP 或 HTTPS 代理。您可以通过修改现有集群的 Proxy 对象或在新集群的 install-config.yaml 文件中配置代理设置，将 OpenShift Container Platform 配置为使用代理。

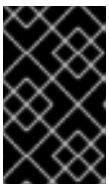
31.1. 先决条件

- 查看 集群需要访问的站点，并确定它们中的任何站点 是否需要绕过代理。默认情况下，所有集群系统的出站流量都需经过代理，包括对托管集群的云供应商 API 的调用。系统范围的代理仅会影响系统组件，而不会影响用户工作负载。若有需要，将站点添加到 Proxy 对象的 spec.noProxy 字段来绕过代理服务器。

注意

对于大多数安装类型，Proxy 对象 status.noProxy 字段使用安装配置中的 networking.machineNetwork[].cidr、networking.clusterNetwork[].cidr 和 networking.serviceNetwork[] 字段的值填充。

对于在 Amazon Web Services(AWS)、Google Cloud Platform(GCP)、Microsoft Azure 和 Red Hat OpenStack Platform(RHOSP)上安装，Proxy 对象 status.noProxy 字段也会使用实例元数据端点填充(169.254.169.254)。



重要

如果您的安装类型不包括设置 networking.machineNetwork[].cidr 字段，则必须在 .status.noProxy 字段中手动包含机器 IP 地址，以确保节点之间的流量可以绕过代理。

31.2. 启用集群范围代理

Proxy 对象用于管理集群范围出口代理。如果在安装或升级集群时没有配置代理，则 Proxy 对象仍会生成，但它会有一个空的 spec。例如：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

集群管理员可以通过修改这个 cluster Proxy 对象来配置 OpenShift Container Platform 的代理。

注意

只支持名为 cluster 的 Proxy 对象，且无法创建额外的代理。

先决条件

- 集群管理员权限

- 已安装 OpenShift Container Platform oc CLI 工具

流程

1. 创建包含代理 HTTPS 连接所需的额外 CA 证书的 ConfigMap。

注意

如果代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名，您可以跳过这一步。

a. 利用以下内容，创建一个名为 user-ca-bundle.yaml 的文件，并提供 PEM 编码证书的值：

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1 这个数据键必须命名为 ca-bundle.crt。

2 一个或多个 PEM 编码的 X.509 证书，用来为代理的身份证书签名。

3 从 Proxy 对象引用的配置映射名称。

4 配置映射必须位于 openshift-config 命名空间中。

b. 从此文件创建配置映射：

```
$ oc create -f user-ca-bundle.yaml
```

2. 使用 oc edit 命令修改 Proxy 对象：

```
$ oc edit proxy/cluster
```

3. 为代理配置所需的字段：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
```

```
- https://www.google.com
trustedCA:
  name: user-ca-bundle 5
```

1. 用于创建集群外 HTTP 连接的代理 URL。URL 方案必须是 http。
2. 用于创建集群外 HTTPS 连接的代理 URL。URL 方案必须是 http 或 https。指定支持 URL 方案的代理的 URL。例如，如果大多数代理被配置为使用 https，则大多数代理都会报告错误，但它们只支持 http。此失败消息可能无法传播到日志，并可能显示为网络连接失败。如果使用侦听来自集群的 https 连接的代理，您可能需要配置集群以接受代理使用的 CA 和证书。
3. 要排除代理的目标域名、域、IP 地址或其他网络 CIDR 的逗号分隔列表。

   在域前面加 . 来仅匹配子域。例如：.y.com 匹配 x.y.com，但不匹配 y.com。使用 * 可对所有目的地绕过所有代理。如果您扩展了未包含在安装配置中 networking.machineNetwork[].cidr 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

   如果未设置 httpProxy 或 httpsProxy 字段，则此字段将被忽略。
4. 将 httpProxy 和 httpsProxy 值写进状态之前，执行就绪度检查时要使用的一个或多个集群外部 URL。
5. 引用 openshift-config 命名空间中的 ConfigMap，其包含代理 HTTPS 连接所需的额外 CA 证书。注意 ConfigMap 必须已经存在，然后才能在这里引用它。此字段是必需的，除非代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名。

4. 保存文件以应用更改。

31.3. 删除集群范围代理服务器

cluster Proxy 对象不能被删除。要从集群中删除代理，请删除 Proxy 对象的所有 spec 字段。

先决条件

- 集群管理员权限
- 已安装 OpenShift Container Platform oc CLI 工具

流程

1. 使用 oc edit 命令来修改代理：

```
$ oc edit proxy/cluster
```

2. 删除 Proxy 对象的所有 spec 字段。例如：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
```

3. **保存文件以使改变生效。**

其他资源

- 替换 CA Bundle 证书
- 代理证书自定义
- 集群范围的代理设置如何应用到 OpenShift Container Platform 节点？

第 32 章 配置自定义 PKI

有些平台组件，如 Web 控制台，使用 Routes 进行通信，且必须信任其他组件的证书与其交互。如果您使用的是自定义公钥基础架构 (PKI)，您必须将其配置为在集群中识别其私有签名的 CA 证书。

您可以使用 Proxy API 添加集群范围的可信 CA 证书。您必须在安装过程中或运行时执行此操作。

- 在 *安装过程* 中，配置集群范围的代理。您需要在 install-config.yaml 文件中的 additionalTrustBundle 设置中定义私有签名的 CA 证书。
  安装程序生成名为 user-ca-bundle 的 ConfigMap，其中包含您定义的附加 CA 证书。然后，Cluster Network Operator 会创建 trusted-ca-bundle ConfigMap，将这些内容与 Red Hat Enterprise Linux CoreOS (RHCOS) 信任捆绑包合并，Proxy 对象的 trustedCA 字段中也会引用此 ConfigMap。

- 在 *运行时*，，修改默认 Proxy 对象使其包含您私有签名的 CA 证书（集群代理启用工作流程的一部分）。这涉及创建包含集群应信任的私有签名 CA 证书的 ConfigMap，然后使用 trustedCA 引用私有签名证书的 ConfigMap 修改代理服务器资源。

注意

安装程序配置的 additionalTrustBundle 字段和 proxy 资源的trustedCA 字段被用来管理集群范围信任捆绑包; 在安装时会使用 additionalTrustBundle ，并在运行时使用代理的trustedCA。

trustedCA 字段是对包含集群组件使用的自定义证书和密钥对的 ConfigMap 的引用。

32.1. 在安装过程中配置集群范围的代理

生产环境可能会拒绝直接访问互联网，而是提供 HTTP 或 HTTPS 代理。您可以通过在 install-config.yaml 文件中配置代理设置，将新的 OpenShift Container Platform 集群配置为使用代理。

先决条件

- 您有一个现有的 install-config.yaml 文件。
- 您检查了集群需要访问的站点，并确定它们中的任何站点是否需要绕过代理。默认情况下，所有集群出口流量都经过代理，包括对托管云供应商 API 的调用。如果需要，您将在 Proxy 对象的 spec.noProxy 字段中添加站点来绕过代理。

注意

Proxy 对象 status.noProxy 字段使用安装配置中的 networking.machineNetwork[].cidr、networking.clusterNetwork[].cidr 和 networking.serviceNetwork[] 字段的值填充。

对于在 Amazon Web Services(AWS)、Google Cloud Platform(GCP)、Microsoft Azure 和 Red Hat OpenStack Platform(RHOSP)上安装，Proxy 对象 status.noProxy 字段也会使用实例元数据端点填充(169.254.169.254)。

流程

1. 编辑 install-config.yaml 文件并添加代理设置。例如：

```
apiVersion: v1
```

```
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: ec2.<aws_region>.amazonaws.com,elasticloadbalancing.
<aws_region>.amazonaws.com,s3.<aws_region>.amazonaws.com 3
additionalTrustBundle: | 4
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
additionalTrustBundlePolicy: <policy_to_add_additionalTrustBundle> 5
```

1. 用于创建集群外 HTTP 连接的代理 URL。URL 方案必须是 http。
2. 用于创建集群外 HTTPS 连接的代理 URL。
3. 要从代理中排除的目标域名、IP 地址或其他网络 CIDR 的逗号分隔列表。在域前面加 . 来仅匹配子域。例如： .y.com 匹配 x.y.com，但不匹配 y.com。使用 * 绕过所有目的地的代理。如果您已将 Amazon EC2、Elastic Load Balancing 和 S3 VPC 端点添加到 VPC 中，您必须将这些端点添加到 noProxy 字段。
4. 如果提供，安装程序会在 openshift-config 命名空间中生成名为 user-ca-bundle 的配置映射，其包含代理 HTTPS 连接所需的一个或多个额外 CA 证书。然后，Cluster Network Operator 会创建 trusted-ca-bundle 配置映射，将这些内容与 Red Hat Enterprise Linux CoreOS（RHCOS）信任捆绑包合并， Proxy 对象的 trustedCA 字段中也会引用此配置映射。additionalTrustBundle 字段是必需的，除非代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名。
5. 可选：决定 Proxy 对象的配置以引用 trustedCA 字段中 user-ca-bundle 配置映射的策略。允许的值是 Proxyonly 和 Always。仅在配置了 http/https 代理时，使用 Proxyonly 引用 user-ca-bundle 配置映射。使用 Always 始终引用 user-ca-bundle 配置映射。默认值为 Proxyonly。

注意

安装程序不支持代理的 readinessEndpoints 字段。

注意

如果安装程序超时，重启并使用安装程序的 wait-for 命令完成部署。例如：

```
$ ./openshift-install wait-for install-complete --log-level debug
```

2. 保存该文件并在安装 OpenShift Container Platform 时引用。

安装程序会创建一个名为 cluster 的集群范围代理，该代理 使用 提供的 install-config.yaml 文件中的代理设置。如果没有提供代理设置，仍然会创建一个 cluster Proxy 对象，但它会有一个空 spec。

注意

只支持名为 cluster 的 Proxy 对象，且无法创建额外的代理。

32.2. 启用集群范围代理

Proxy 对象用于管理集群范围出口代理。如果在安装或升级集群时没有配置代理，则 Proxy 对象仍会生成，但它会有一个空的 spec。例如：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

集群管理员可以通过修改这个 cluster Proxy 对象来配置 OpenShift Container Platform 的代理。

注意

只支持名为 cluster 的 Proxy 对象，且无法创建额外的代理。

先决条件

- 集群管理员权限
- 已安装 OpenShift Container Platform oc CLI 工具

流程

1. 创建包含代理 HTTPS 连接所需的额外 CA 证书的 ConfigMap。

注意

如果代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名，您可以跳过这一步。

a. 利用以下内容，创建一个名为 user-ca-bundle.yaml 的文件，并提供 PEM 编码证书的值：

```
apiVersion: v1
data:
  ca-bundle.crt: | (1)
    <MY_PEM_ENCODED_CERTS> (2)
kind: ConfigMap
metadata:
  name: user-ca-bundle (3)
  namespace: openshift-config (4)
```

(1) 这个数据键必须命名为 ca-bundle.crt。

(2) 一个或多个 PEM 编码的 X.509 证书，用来为代理的身份证书签名。

(3) 从 Proxy 对象引用的配置映射名称。

(4) 配置映射必须位于 openshift-config 命名空间中。

b. 从此文件创建配置映射：

```
$ oc create -f user-ca-bundle.yaml
```

2. 使用 oc edit 命令修改 Proxy 对象：

```
$ oc edit proxy/cluster
```

3. 为代理配置所需的字段：

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1 用于创建集群外 HTTP 连接的代理 URL。URL 方案必须是 http。

2 用于创建集群外 HTTPS 连接的代理 URL。URL 方案必须是 http 或 https。指定支持 URL 方案的代理的 URL。例如，如果大多数代理被配置为使用 https，则大多数代理都会报告错误，但它们只支持 http。此失败消息可能无法传播到日志，并可能显示为网络连接失败。如果使用侦听来自集群的 https 连接的代理，您可能需要配置集群以接受代理使用的 CA 和证书。

3 要排除代理的目标域名、域、IP 地址或其他网络 CIDR 的逗号分隔列表。

在域前面加 . 来仅匹配子域。例如： .y.com 匹配 x.y.com，但不匹配 y.com。使用 * 可对所有目的地绕过所有代理。如果您扩展了未包含在安装配置中 networking.machineNetwork[].cidr 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

如果未设置 httpProxy 或 httpsProxy 字段，则此字段将被忽略。

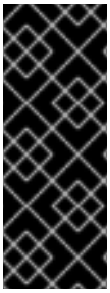
4 将 httpProxy 和 httpsProxy 值写进状态之前，执行就绪度检查时要使用的一个或多个集群外部 URL。

5 引用 openshift-config 命名空间中的 ConfigMap，其包含代理 HTTPS 连接所需的额外 CA 证书。注意 ConfigMap 必须已经存在，然后才能在这里引用它。此字段是必需的，除非代理的身份证书由来自 RHCOS 信任捆绑包的颁发机构签名。

4. 保存文件以使改变生效。

32.3. 使用 OPERATOR 进行证书注入

在您的自定义 CA 证书通过 ConfigMap 添加到集群中后，Cluster Network Operator 会将用户提供的证书和系统 CA 证书合并到单一捆绑包中，并将合并的捆绑包注入请求信任捆绑包注入的 Operator。



重要

在配置映射中添加 config.openshift.io/inject-trusted-cabundle="true" 标签后，会删除其中的现有数据。Cluster Network Operator 获取配置映射的所有权，并只接受 ca-bundle 作为数据。您必须使用单独的配置映射存储 service-ca.crt，方法是使用 service.beta.openshift.io/inject-cabundle=true 注解或类似的配置。在同一配置映射中添加 config.openshift.io/inject-trusted-cabundle="true" 标签和 service.beta.openshift.io/inject-cabundle=true 注解可能会导致问题。

Operator 通过创建一个带有以下标签的空 ConfigMap 来请求此注入：

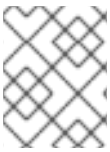
```
config.openshift.io/inject-trusted-cabundle="true"
```

空 ConfigMap 示例：

```
apiVersion: v1
data: {}
kind: ConfigMap
metadata:
  labels:
    config.openshift.io/inject-trusted-cabundle: "true"
  name: ca-inject (1)
  namespace: apache
```

1 指定空 ConfigMap 名称。

Operator 将这个 ConfigMap 挂载到容器的本地信任存储中。



注意

只有在 Red Hat Enterprise Linux CoreOS (RHCOS) 信任捆绑包中没有包括证书时才需要添加可信的 CA 证书。

证书注入不仅限于 Operator。当使用 config.openshift.io/inject-trusted-cabundle=true标记（label）创建一个空的 ConfigMap 时，Cluster Network Operator会跨命名空间注入证书 。

ConfigMap 可以驻留在任何命名空间中，但 ConfigMap 必须作为卷挂载到需要自定义 CA 的 Pod 中的每个容器。例如：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
```

```
    volumeMounts:
    - name: trusted-ca
      mountPath: /etc/pki/ca-trust/extracted/pem
      readOnly: true
  volumes:
  - name: trusted-ca
    configMap:
      name: trusted-ca
      items:
        - key: ca-bundle.crt 1
          path: tls-ca-bundle.pem 2
```

1 CA-bundle.crt 需要作为 ConfigMap 密钥。

2 TLS-ca-bundle.pem 需要作为 ConfigMap 的路径。

第 33 章 RHOSP 负载均衡

33.1. 负载均衡器服务的限制

Red Hat OpenStack Platform (RHOSP) 上的 OpenShift Container Platform 集群使用 Octavia 来处理负载均衡器服务。因此，此类集群有很多功能限制。

RHOSP Octavia 有两个支持的供应商： Amphora 和 OVN。这些供应商在可用功能以及实施详情方面有所不同。这些差异会影响在集群中创建的负载均衡器服务。

33.1.1. 本地外部流量策略

您可以在负载均衡器服务上设置外部流量策略 (ETP) 参数 **.spec.externalTrafficPolicy**，以在到达服务端点 pod 时保留传入流量的源 IP 地址。但是，如果您的集群使用 Amphora Octavia 供应商，流量的源 IP 将替换为 Amphora 虚拟机的 IP 地址。如果您的集群使用 OVN Octavia 供应商，则不会发生此行为。

将 **ETP** 选项设置为 **Local** 需要为负载均衡器创建运行状况监控器。如果没有健康监控器，流量可以路由到没有功能端点的节点，这会导致连接丢弃。要强制 Cloud Provider OpenStack 创建运行状况监视器，您必须将云供应商配置中的 **create-monitor** 选项的值设置为 **true**。

在 RHOSP 16.2 中，OVN Octavia 供应商不支持健康监控器。因此，不支持将 ETP 设置为 local。

在 RHOSP 16.2 中，Amphora Octavia 供应商不支持 UDP 池中的 HTTP 监视器。因此，UDP 负载均衡器服务会改为创建 **UDP-CONNECT** 监视器。由于实现详情，此配置只能使用 OVN-Kubernetes CNI 插件正常工作。当使用 OpenShift SDN CNI 插件时，UDP 服务会变得不可靠。这个问题也会影响任何 RHOSP 版本中的 OVN Octavia 供应商，因为驱动程序不支持 HTTP 健康监控器。

33.2. 使用 OCTAVIA 为应用程序流量扩展集群

在 Red Hat OpenStack Platform（RHOSP）上运行的 OpenShift Container Platform 集群可以使用 Octavia 负载均衡服务在多个虚拟机（VM）或浮动 IP 地址间分配流量。这个功能减少了单一机器或地址生成的瓶颈。

您必须创建自己的 Octavia 负载均衡器，将其用于应用程序网络扩展。

33.2.1. 使用 Octavia 扩展集群

如果要使用多个 API 负载均衡器，请创建一个 Octavia 负载均衡器，然后将集群配置为使用它。

先决条件

- Octavia 包括在您的 Red Hat OpenStack Platform（RHOSP）部署中。

流程

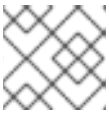
1. 在命令行中创建一个使用 Amphora 驱动程序的 Octavia 负载均衡器：

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

   可以使用自己选择的名称而不是 **API_OCP_CLUSTER**。

2. 负载均衡器成为活跃后，创建监听程序：

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```



注意

要查看负载均衡器的状态，请输入 **openstack loadbalancer list**。

3. 创建一个使用轮循算法的池，并启用了会话持久性：

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

4. 为确保 control plane 机器可用，创建一个健康监控器：

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --
type TCP API_OCP_CLUSTER_pool_6443
```

5. 将 control plane 机器作为负载均衡器池的成员添加：

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER  --protocol-port 6443
API_OCP_CLUSTER_pool_6443
done
```

6. 可选：要重复使用集群 API 浮动 IP 地址，取消设置它：

```
$ openstack floating ip unset $API_FIP
```

7. 为创建的负载均衡器 VIP 添加未设置的 API_FIP 或一个新地址：

```
$ openstack floating ip set  --port $(openstack loadbalancer show -c <vip_port_id> -f
value API_OCP_CLUSTER) $API_FIP
```

您的集群现在使用 Octavia 进行负载平衡。

33.3. 用户管理的负载均衡器的服务

您可以在 Red Hat OpenStack Platform（RHOSP）上配置 OpenShift Container Platform 集群，使其外部管理的负载均衡器来代替默认负载均衡器。

重要

配置用户管理的负载均衡器取决于您的厂商的负载均衡器。

本节中的信息和示例仅用于指导目的。有关供应商负载均衡器的更多信息，请参阅供应商文档。

红帽支持用户管理的负载均衡器的以下服务：

- Ingress Controller
- OpenShift API
- OpenShift MachineConfig API

您可以选择是否要为用户管理的负载均衡器配置一个或多个所有服务。仅配置 Ingress Controller 服务是一个通用的配置选项。要更好地了解每个服务，请查看以下图表：

图 33.1. 显示 OpenShift Container Platform 环境中运行的 Ingress Controller 的网络工作流示例

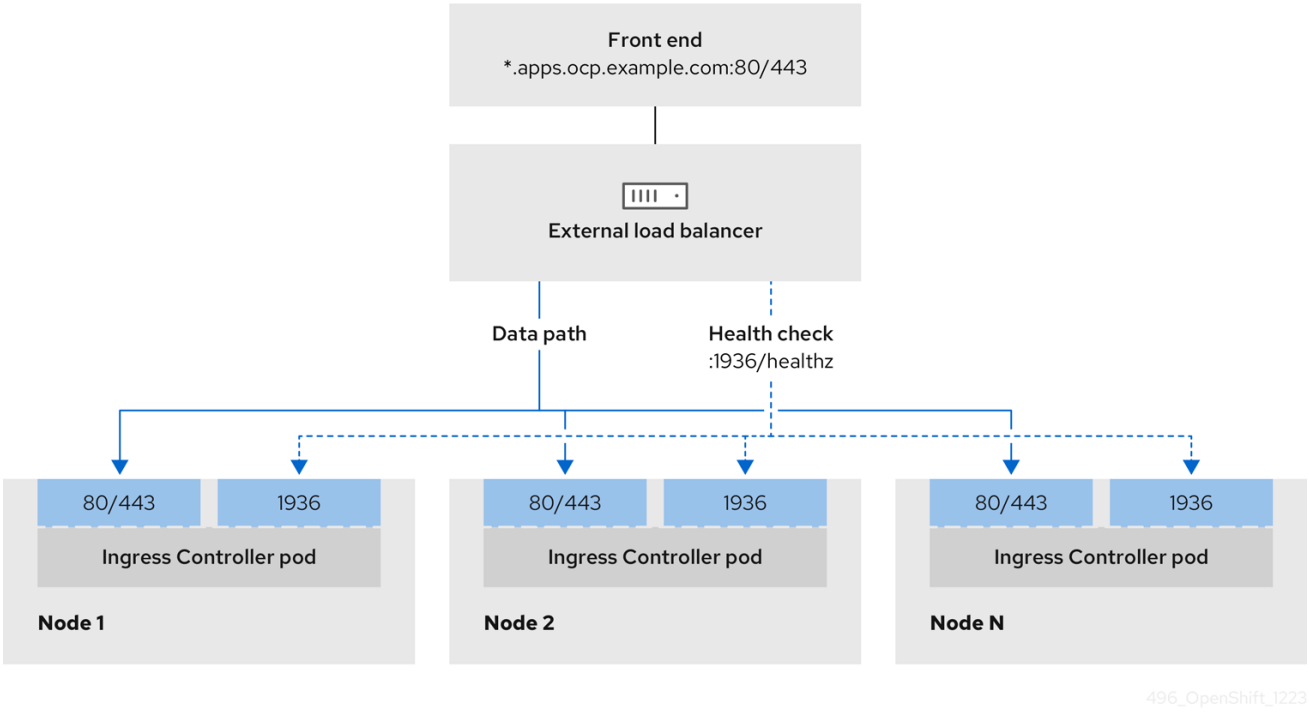


图 33.2. 显示 OpenShift Container Platform 环境中运行的 OpenShift API 的网络工作流示例

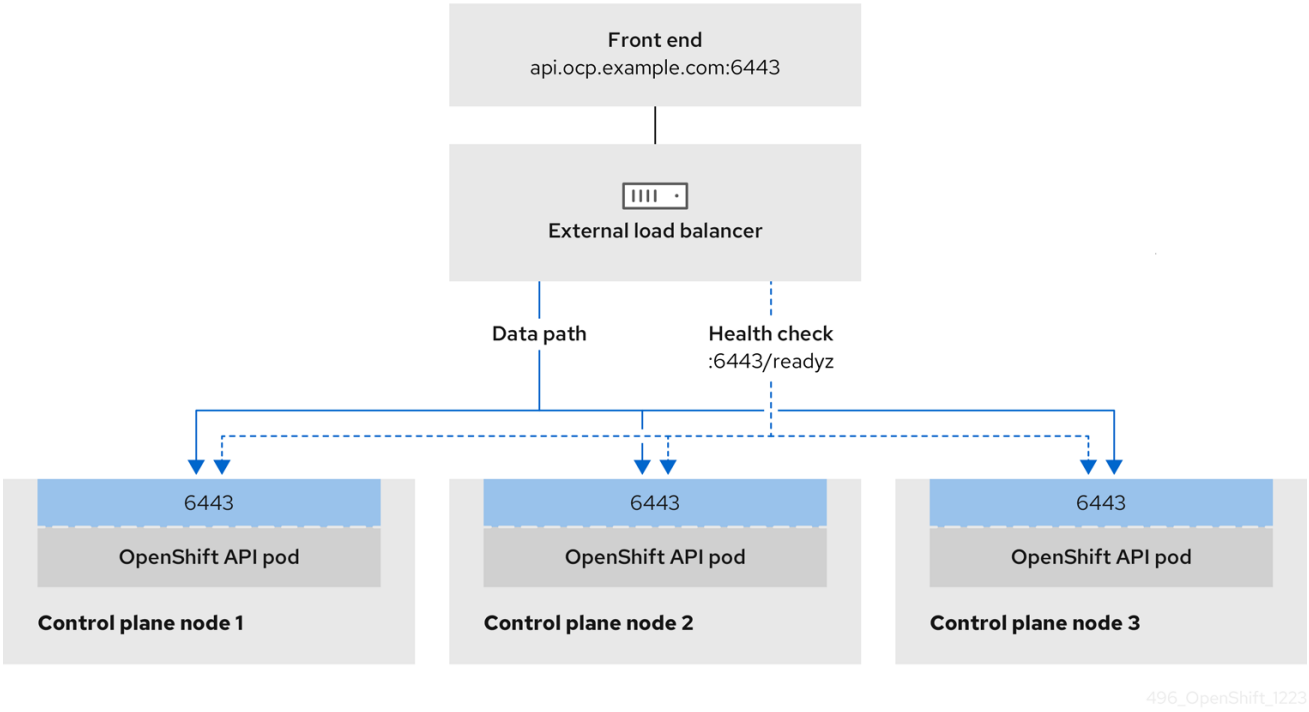
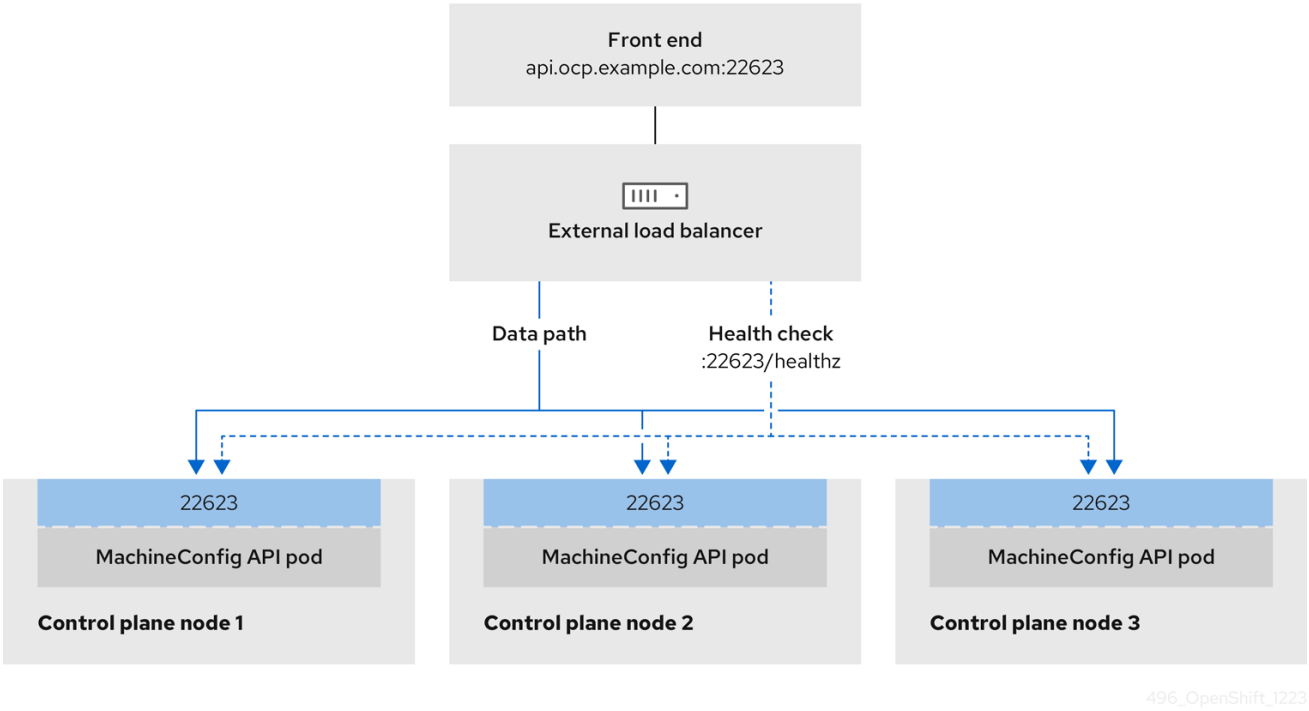


图 33.3. 显示 OpenShift Container Platform 环境中运行的 OpenShift MachineConfig API 的网络工作流示例



用户管理的负载均衡器支持以下配置选项：

- 使用节点选择器将 Ingress Controller 映射到一组特定的节点。您必须为这个集合中的每个节点分配一个静态 IP 地址，或者将每个节点配置为从动态主机配置协议(DHCP)接收相同的 IP 地址。基础架构节点通常接收这种类型的配置。
- 以子网上的所有 IP 地址为目标。此配置可减少维护开销，因为您可以在这些网络中创建和销毁节点，而无需重新配置负载均衡器目标。如果您使用较小的网络上的机器集来部署入口 pod，如 /27 或 /28，您可以简化负载均衡器目标。

  提示

  您可以通过检查机器配置池的资源来列出网络中存在的所有 IP 地址。

在为 OpenShift Container Platform 集群配置用户管理的负载均衡器前，请考虑以下信息：

- 对于前端 IP 地址，您可以对前端 IP 地址、Ingress Controller 的负载均衡器和 API 负载均衡器使用相同的 IP 地址。查看厂商的文档以获取此功能的相关信息。
- 对于后端 IP 地址，请确保 OpenShift Container Platform control plane 节点的 IP 地址在用户管理的负载均衡器生命周期内不会改变。您可以通过完成以下操作之一来实现此目的：
  - 为每个 control plane 节点分配一个静态 IP 地址。
  - 将每个节点配置为在每次节点请求 DHCP 租期时从 DHCP 接收相同的 IP 地址。根据供应商，DHCP 租期可能采用 IP 保留或静态 DHCP 分配的形式。
- 在 Ingress Controller 后端服务的用户管理的负载均衡器中手动定义运行 Ingress Controller 的每个节点。例如，如果 Ingress Controller 移到未定义节点，则可能会出现连接中断。

33.3.1. 配置用户管理的负载均衡器

您可以在 Red Hat OpenStack Platform（RHOSP）上配置 OpenShift Container Platform 集群，使其外部管理的负载均衡器来代替默认负载均衡器。

重要

在配置用户管理的负载均衡器前，请确保阅读用户管理的负载均衡器部分。

阅读适用于您要为用户管理的负载均衡器配置的服务的以下先决条件。

注意

MetalLB，在集群中运行，充当用户管理的负载均衡器。

OpenShift API 的先决条件

- 您定义了前端 IP 地址。
- TCP 端口 6443 和 22623 在负载均衡器的前端 IP 地址上公开。检查以下项：
  - 端口 6443 提供对 OpenShift API 服务的访问。
  - 端口 22623 可以为节点提供 ignition 启动配置。
- 前端 IP 地址和端口 6443 可以被您的系统的所有用户访问，其位置为 OpenShift Container Platform 集群外部。
- 前端 IP 地址和端口 22623 只能被 OpenShift Container Platform 节点访问。
- 负载均衡器后端可以在端口 6443 和 22623 上与 OpenShift Container Platform control plane 节点通信。

Ingress Controller 的先决条件

- 您定义了前端 IP 地址。
- TCP 端口 443 和 80 在负载均衡器的前端 IP 地址上公开。
- 前端 IP 地址、端口 80 和端口 443 可以被您的系统所有用户访问，以及 OpenShift Container Platform 集群外部的位置。
- 前端 IP 地址、端口 80 和端口 443 可被 OpenShift Container Platform 集群中运行的所有节点访问。
- 负载均衡器后端可以在端口 80、443 和 1936 上与运行 Ingress Controller 的 OpenShift Container Platform 节点通信。

健康检查 URL 规格的先决条件

您可以通过设置健康检查 URL 来配置大多数负载均衡器，以确定服务是否可用或不可用。OpenShift Container Platform 为 OpenShift API、Machine Configuration API 和 Ingress Controller 后端服务提供这些健康检查。

以下示例显示了之前列出的后端服务的健康检查规格：

Kubernetes API 健康检查规格示例

```
Path: HTTPS:6443/readyz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Machine Config API 健康检查规格示例

```
Path: HTTPS:22623/healthz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Ingress Controller 健康检查规格示例

```
Path: HTTP:1936/healthz/ready
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 5
Interval: 10
```

流程

1. 配置 HAProxy Ingress Controller，以便您可以在端口 6443、22623、443 和 80 上从负载均衡器访问集群。根据您的需要，您可以在 HAProxy 配置中指定来自多个子网的单个子网或 IP 地址的 IP 地址。

   带有列出子网的 HAProxy 配置示例

```
# ...
listen my-cluster-api-6443
    bind 192.168.1.100:6443
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /readyz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:6443 check inter 10s rise 2 fall 2

listen my-cluster-machine-config-api-22623
    bind 192.168.1.100:22623
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:22623 check inter 10s rise 2 fall 2
```

```
listen my-cluster-apps-443
    bind 192.168.1.100:443
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz/ready
  http-check expect status 200
    server my-cluster-worker-0 192.168.1.111:443 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-1 192.168.1.112:443 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-2 192.168.1.113:443 check port 1936 inter 10s rise 2 fall 2

listen my-cluster-apps-80
   bind 192.168.1.100:80
   mode tcp
   balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz/ready
  http-check expect status 200
    server my-cluster-worker-0 192.168.1.111:80 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-1 192.168.1.112:80 check port 1936 inter 10s rise 2 fall 2
    server my-cluster-worker-2 192.168.1.113:80 check port 1936 inter 10s rise 2 fall 2
# ...
```

带有多个列出子网的 HAProxy 配置示例

```
# ...
listen api-server-6443
    bind *:6443
    mode tcp
      server master-00 192.168.83.89:6443 check inter 1s
      server master-01 192.168.84.90:6443 check inter 1s
      server master-02 192.168.85.99:6443 check inter 1s
      server bootstrap 192.168.80.89:6443 check inter 1s

listen machine-config-server-22623
    bind *:22623
    mode tcp
      server master-00 192.168.83.89:22623 check inter 1s
      server master-01 192.168.84.90:22623 check inter 1s
      server master-02 192.168.85.99:22623 check inter 1s
      server bootstrap 192.168.80.89:22623 check inter 1s

listen ingress-router-80
    bind *:80
    mode tcp
    balance source
      server worker-00 192.168.83.100:80 check inter 1s
      server worker-01 192.168.83.101:80 check inter 1s

listen ingress-router-443
    bind *:443
    mode tcp
    balance source
```

```
    server worker-00 192.168.83.100:443 check inter 1s
    server worker-01 192.168.83.101:443 check inter 1s

listen ironic-api-6385
  bind *:6385
  mode tcp
  balance source
    server master-00 192.168.83.89:6385 check inter 1s
    server master-01 192.168.84.90:6385 check inter 1s
    server master-02 192.168.85.99:6385 check inter 1s
    server bootstrap 192.168.80.89:6385 check inter 1s

listen inspector-api-5050
  bind *:5050
  mode tcp
  balance source
    server master-00 192.168.83.89:5050 check inter 1s
    server master-01 192.168.84.90:5050 check inter 1s
    server master-02 192.168.85.99:5050 check inter 1s
    server bootstrap 192.168.80.89:5050 check inter 1s
# ...
```

2. 使用 curl CLI 命令验证用户管理的负载均衡器及其资源是否正常运行：

   a. 运行以下命令并查看响应，验证集群机器配置 API 是否可以被 Kubernetes API 服务器资源访问：

   ```
   $ curl https://<loadbalancer_ip_address>:6443/version --insecure
   ```

   如果配置正确，您会收到 JSON 对象的响应：

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
     "gitCommit": "ad103ed",
     "gitTreeState": "clean",
     "buildDate": "2019-01-09T06:44:10Z",
     "goVersion": "go1.10.3",
     "compiler": "gc",
     "platform": "linux/amd64"
   }
   ```

   b. 运行以下命令并观察输出，验证集群机器配置 API 是否可以被 Machine 配置服务器资源访问：

   ```
   $ curl -v https://<loadbalancer_ip_address>:22623/healthz --insecure
   ```

   如果配置正确，命令的输出会显示以下响应：

   ```
   HTTP/1.1 200 OK
   Content-Length: 0
   ```

c. 运行以下命令并观察输出，验证控制器是否可以被端口 80 上的 Ingress Controller 资源访问：

```
$ curl -I -L -H "Host: console-openshift-console.apps.<cluster_name>.
<base_domain>" http://<load_balancer_front_end_IP_address>
```

如果配置正确，命令的输出会显示以下响应：

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.ocp4.private.opequon.net/
cache-control: no-cache
```

d. 运行以下命令并观察输出，验证控制器是否可以被端口 443 上的 Ingress Controller 资源访问：

```
$ curl -I -L --insecure --resolve console-openshift-console.apps.<cluster_name>.
<base_domain>:443:<Load Balancer Front End IP Address> https://console-
openshift-console.apps.<cluster_name>.<base_domain>
```

如果配置正确，命令的输出会显示以下响应：

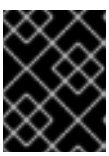
```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja2
61dGLgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673;
path=/; HttpOnly; Secure; SameSite=None
cache-control: private
```

3. 配置集群的 DNS 记录，使其以用户管理的负载均衡器的前端 IP 地址为目标。您必须在负载均衡器上将记录更新为集群 API 和应用程序的 DNS 服务器。

修改 DNS 记录示例

```
<load_balancer_ip_address>  A  api.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

```
<load_balancer_ip_address>  A apps.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```



重要

DNS 传播可能需要一些时间才能获得每个 DNS 记录。在验证每个记录前，请确保每个 DNS 记录传播。

4. 要使 OpenShift Container Platform 集群使用用户管理的负载均衡器，您必须在集群的 install-config.yaml 文件中指定以下配置：

```
# ...
platform:
  openstack:
    loadBalancer:
      type: UserManaged (1)
      apiVIPs:
      - <api_ip> (2)
      ingressVIPs:
      - <ingress_ip> (3)
# ...
```

(1) 为 type 参数设置 UserManaged，为集群指定用户管理的负载均衡器。参数默认为 OpenShiftManagedDefault，它表示默认的内部负载均衡器。对于 openshift-kni-infra 命名空间中定义的服务，用户管理的负载均衡器可将 coredns 服务部署到集群中的 pod，但忽略 keepalived 和 haproxy 服务。

(2) 指定用户管理的负载均衡器时所需的参数。指定用户管理的负载均衡器的公共 IP 地址，以便 Kubernetes API 可以与用户管理的负载均衡器通信。

(3) 指定用户管理的负载均衡器时所需的参数。指定用户管理的负载均衡器的公共 IP 地址，以便用户管理的负载均衡器可以管理集群的入口流量。

验证

1. 使用 curl CLI 命令验证用户管理的负载均衡器和 DNS 记录配置是否正常工作：

   a. 运行以下命令并查看输出，验证您可以访问集群 API：

   ```
   $ curl https://api.<cluster_name>.<base_domain>:6443/version --insecure
   ```

   如果配置正确，您会收到 JSON 对象的响应：

   ```
   {
     "major": "1",
     "minor": "11+",
     "gitVersion": "v1.11.0+ad103ed",
     "gitCommit": "ad103ed",
     "gitTreeState": "clean",
     "buildDate": "2019-01-09T06:44:10Z",
     "goVersion": "go1.10.3",
     "compiler": "gc",
     "platform": "linux/amd64"
     }
   ```

   b. 运行以下命令并查看输出，验证您可以访问集群机器配置：

   ```
   $ curl -v https://api.<cluster_name>.<base_domain>:22623/healthz --insecure
   ```

   如果配置正确，命令的输出会显示以下响应：

```
HTTP/1.1 200 OK
Content-Length: 0
```

c. 运行以下命令并查看输出，验证您可以在端口上访问每个集群应用程序：

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L
--insecure
```

如果配置正确，命令的输出会显示以下响应：

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/
cache-control: no-cacheHTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=39HoZgztDnzjJkq/JuLJMeoKNXlfiVv2YgZc09c3TBOBU4Nl6kDXaJH1LdicNh
N1UsQWzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be;
path=/; HttpOnly; Secure; SameSite=None
cache-control: private
```

d. 运行以下命令并查看输出，验证您可以在端口 443 上访问每个集群应用程序：

```
$ curl https://console-openshift-console.apps.<cluster_name>.<base_domain> -I -
L --insecure
```

如果配置正确，命令的输出会显示以下响应：

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja2
61dGLgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673;
path=/; HttpOnly; Secure; SameSite=None
cache-control: private
```

第 34 章 使用 METALLB 进行负载平衡

34.1. 关于 METALLB 和 METALLB OPERATOR

作为集群管理员，您可以将 MetalLB Operator 添加到集群中，以便在将 LoadBalancer 类型服务添加到集群中时，MetalLB 可为该服务添加外部 IP 地址。外部 IP 地址添加到集群的主机网络中。

34.1.1. 何时使用 MetalLB

当您有裸机集群或类似裸机的基础架构时，使用 MetalLB 有价值，并且您希望通过外部 IP 地址对应用程序进行容错访问。

您必须配置网络基础架构，以确保外部 IP 地址的网络流量从客户端路由到集群的主机网络。

使用 MetalLB Operator 部署 MetalLB 后，当添加类型为 LoadBalancer 的服务时，MetalLB 提供了一个平台原生负载均衡器。

在 layer2 模式中的 MetalLB 操作通过使用与 IP 故障转移类似的机制提供对故障切换的支持。但是，MetalLB 利用基于 gosip 的协议来识别节点故障实例，而不依赖于虚拟路由器冗余协议 (VRRP) 和 keepalived。当检测到故障转移时，另一个节点会假定领导节点的角色，并且分配了一个 gratuitous ARP 消息来广播此更改。

MetalLB 在 layer3 或边框网关协议 (BGP) 模式下操作，将故障检测委派给网络。OpenShift Container Platform 节点建立连接的 BGP 路由器或路由器将识别任何节点故障并终止到该节点的路由。

最好使用 MetalLB 而不是 IP 故障转移来确保 pod 和服务的高可用性。

34.1.2. MetalLB Operator 自定义资源

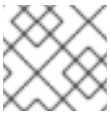
MetalLB Operator 为以下自定义资源监控自己的命名空间：

MetalLB

当您在集群中添加 MetalLB 自定义资源时，MetalLB Operator 会在集群中部署 MetalLB。Operator 只支持单个自定义资源实例。如果删除了实例，Operator 会从集群中删除 MetalLB。

IPAddressPool

MetalLB 需要一个或多个 IP 地址池，您可以在添加类型为 LoadBalancer 的服务时分配给服务。一个 IPAddressPool，包含 IP 地址列表。列表可以是使用范围设置的单个 IP 地址，如 1.1.1.1-1.1.1.1、以 CIDR 表示法指定的范围、指定为起始和结束地址的范围，或者以连字符分隔的、两者的组合。IPAddressPool 需要一个名称。文档使用 doc-example、doc-example-reserved 和 doc-example-ipv6 等名称。MetalLB 控制器 从 IPAddressPool 中的地址池中分配 IP 地址。L2Advertisement 和 BGPAdvertisement 自定义资源启用从一个指定池中广告一个给定 IP。您可以使用 IPAddressPool 自定义资源中的 spec.serviceAllocation 规格将 IP 地址从 IPAddressPool 分配给服务和命名空间。



注意

单个 IPAddressPool 可以被 L2 公告和 BGP 公告来引用。

BGPPeer

BGP peer 自定义资源标识 MetalLB 进行通信的 BGP 路由器、路由器的 AS 数量、MetalLB 的 AS 编号，以及路由公告的自定义。MetalLB 将服务负载平衡器 IP 地址的路由公告给一个或多个 BGP 对等点。

BFDProfile

BFD 配置集自定义资源可为 BGP peer 配置双向转发检测(BFD)。BFD 提供比 BGP 单独提供的路径故障检测速度。

L2Advertisement

L2Advertisement 自定义资源使用 L2 协议广告一个来自 IPAddressPool 的 IP。

BGPAdvertisement

BGPAdvertisement 自定义资源使用 BGP 协议广告一个来自 IPAddressPool 的 IP。

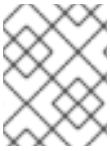
在将 MetalLB 自定义资源添加到集群，且 Operator 部署了 MetalLB 后，controller 和 speaker MetalLB 软件组件将开始运行。

MetalLB 验证所有相关自定义资源。

34.1.3. MetalLB 软件组件

安装 MetalLB Operator 时，metallb-operator-controller-manager 部署会启动一个 pod。pod 是 Operator 的实施。pod 监控所有相关资源的更改。

当 Operator 启动 MetalLB 实例时，它会启动一个 controller 部署和一个 speaker 守护进程集。



注意

您可以在 MetalLB 自定义资源中配置部署规格，以管理 controller 和 speaker pod 如何在集群中部署和运行。有关这些部署规格的更多信息，请参阅*附加资源部分*。

controller

Operator 会启动部署和单个 pod。当您添加类型为 LoadBalancer 的服务时，Kubernetes 使用 controller 从地址池中分配 IP 地址。如果服务失败，请验证 controller pod 日志中有以下条目：

输出示例

```
"event":"ipAllocated","ip":"172.22.0.201","msg":"IP address assigned by controller
```

speaker

Operator 为 speaker pod 启动守护进程集。默认情况下，在集群的每个节点上启动 pod。您可以在启动 MetalLB 时在 MetalLB 自定义资源中指定节点选择器，将 pod 限制到特定的节点。如果 controller 为服务分配了 IP 地址，并且服务仍不可用，请阅读 speaker pod 日志。如果 speaker pod 不可用，请运行 oc describe pod -n 命令。

对于第 2 层模式，控制器 为服务分配 IP 地址后，speaker pod 使用一种算法来确定哪些 speaker pod 将宣布负载均衡器 IP 地址。该算法涉及对节点名称和负载均衡器 IP 地址进行哈希处理。如需更多信息，请参阅"MetalLB 和外部流量策略"。speaker 使用地址解析协议 (ARP) 来宣布 IPv4 地址和邻居发现协议 (NDP) 来宣布 IPv6 地址。

对于 Border Gateway Protocol (BGP) 模式，controller 为服务分配 IP 地址后，每个 speaker pod 为其 BGP 对等点公告负载均衡器 IP 地址。您可以配置节点在 BGP 对等点上启动 BGP 会话。

对负载均衡器 IP 地址的请求会路由到具有声明 IP 地址的 speaker 的节点。节点接收数据包后，服务代理会将数据包路由到该服务的端点。在最佳情况下，端点可以位于同一节点上，也可以位于另一节点上。每次建立连接时，服务代理都会选择一个端点。

34.1.4. MetalLB 和外部流量策略

使用第 2 层模式时，集群中的一个节点会接收服务 IP 地址的所有流量。使用 BGP 模式时，主机网络上的路由器会打开与集群中其中一个节点的连接，用于新客户端连接。集群在进入节点后如何处理流量受外部流量策略的影响。

cluster

这是 **spec.externalTrafficPolicy** 的默认值。
使用 **cluster** 流量策略时，节点接收流量后，服务代理会将流量分发到服务中的所有容器集。此策略在 pod 之间提供统一流量分布，但它会模糊客户端 IP 地址，并可能会在 pod 中显示流量源自节点而不是客户端的应用。

local

采用 **local** 流量策略时，节点接收流量后，服务代理仅将流量发送到同一节点上的 pod。例如，如果节点上的 **speaker** pod 宣布外部服务 IP，则所有流量都发送到节点 A。流量进入节点 A 后，服务代理仅将流量发送到节点 A 上的服务的 pod。位于其他节点上的服务的 Pod 不会从节点 A 接收任何流量。在需要故障转移时，其他节点上的服务的 Pod 充当副本。
此策略不会影响客户端 IP 地址。应用容器集可以确定来自传入连接的客户端 IP 地址。

注意

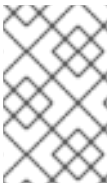
在 BGP 模式中配置外部流量策略时，以下信息非常重要。

虽然 MetalLB 公告来自所有有资格的节点的负载均衡器 IP 地址，但可能会限制在路由器的容量下，以建立同等成本多路径(ECMP)路由。如果广告 IP 的节点数量大于路由器的 ECMP 组的限制，路由器将使用比广告 IP 的节点数量少的节点。

例如，如果外部流量策略设置为 **local**，且路由器将 ECMP 组限制设置为 16，实施 LoadBalancer 服务的 pod 部署在 30 个节点上，这会导致在 14 个节点上部署的 pod 不接收任何流量。在这种情况下，最好将该服务的外部流量策略设置为 **cluster**。

34.1.5. 第 2 层模式的 MetalLB 概念

在第 2 层模式中，一个节点上的 **speaker** pod 向主机网络宣布服务的外部 IP 地址。从网络的角度来看，节点似乎有多个 IP 地址分配给网络接口。



注意

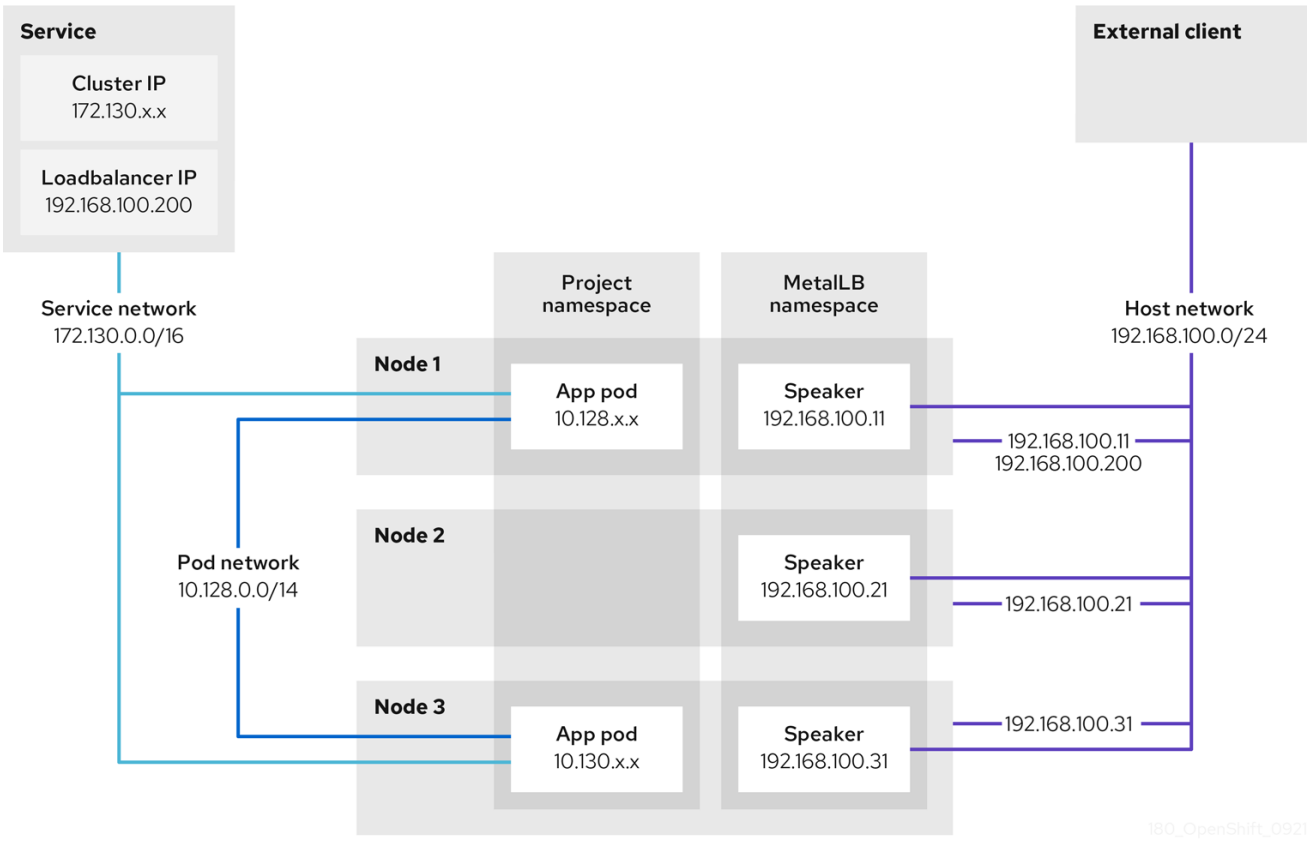
在第 2 层模式中，MetalLB 依赖于 ARP 和 NDP。这些协议在特定子网中实施本地地址解析。在这种情况下，客户端必须能够访问由 MetalLB 分配的 VIP，它与节点位于同一个子网中，以便 MetalLB 正常工作。

speaker pod 响应 IPv4 服务和 IPv6 的 NDP 请求。

在第 2 层模式中，服务 IP 地址的所有流量都通过一个节点进行路由。在流量进入节点后，CNI 网络供应商的服务代理会将流量分发到该服务的所有 pod。

由于服务的所有流量都通过第 2 层模式中的单一节点进入，所以严格意义上，MetalLB 不会为第 2 层实施负载平衡器。相反，MetalLB 为第 2 层实施故障转移机制，以便在 **speaker** pod 不可用时，不同节点上的 **speaker** pod 可以宣布服务 IP 地址。

当节点不可用时，自动故障转移。其他节点上的 **speaker** pod 检测到节点不可用，新的 **speaker** pod 和节点从故障节点上拥有服务 IP 地址的所有权。



上图显示了与 MetalLB 相关的以下概念：

- 应用程序可以通过在 172.130.0.0/16 子网上具有集群 IP 的服务获取。该 IP 地址可以从集群内部访问。服务也有一个外部 IP 地址，用于分配给服务的 MetalLB，即 192.168.100.200。
- 节点 1 和 3 具有应用程序的 pod。
- speaker 守护进程集在每个节点上运行一个 pod。MetalLB Operator 启动这些 pod。
- 每个 speaker pod 都是主机网络的 pod。容器集的 IP 地址与主机网络上节点的 IP 地址相同。
- 节点 1 上的 speaker pod 使用 ARP 声明服务的外部 IP 地址192.168.100.200。声明外部 IP 地址的 speaker pod 必须与服务的端点位于同一个节点上，端点必须为 Ready 条件。
- 客户端流量路由到主机网络，并连接到 192.168.100.200 IP 地址。在流量进入节点后，服务代理会根据您为服务设置的外部流量策略，将流量发送到同一节点上的应用 pod 或其他节点。
  - 如果服务的外部流量策略设置为 cluster，则会从运行 speaker pod 的节点选择广告 192.168.100.200 负载均衡器 IP 地址的节点。只有该节点才能接收该服务的流量。
  - 如果服务的外部流量策略设置为 local，则会从运行 speaker pod 的节点以及至少一个服务的端点选择广告 192.168.100.200 负载均衡器 IP 地址的节点。只有该节点才能接收该服务的流量。在上图中，节点 1 或 3 将广告 192.168.100.200。
- 如果节点 1 不可用，则外部 IP 地址将故障转移到另一节点。在具有应用 pod 和服务端点实例的另一个节点上，speaker Pod 开始宣布外部 IP 地址 192.168.100.200，新节点接收客户端流量。在图中，唯一的候选项是节点 3。

34.1.6. BGP 模式的 MetalLB 概念

在 BGP 模式中，默认情况下每个 speaker pod 都会向每个 BGP 对等广告一个服务的负载均衡器 IP 地

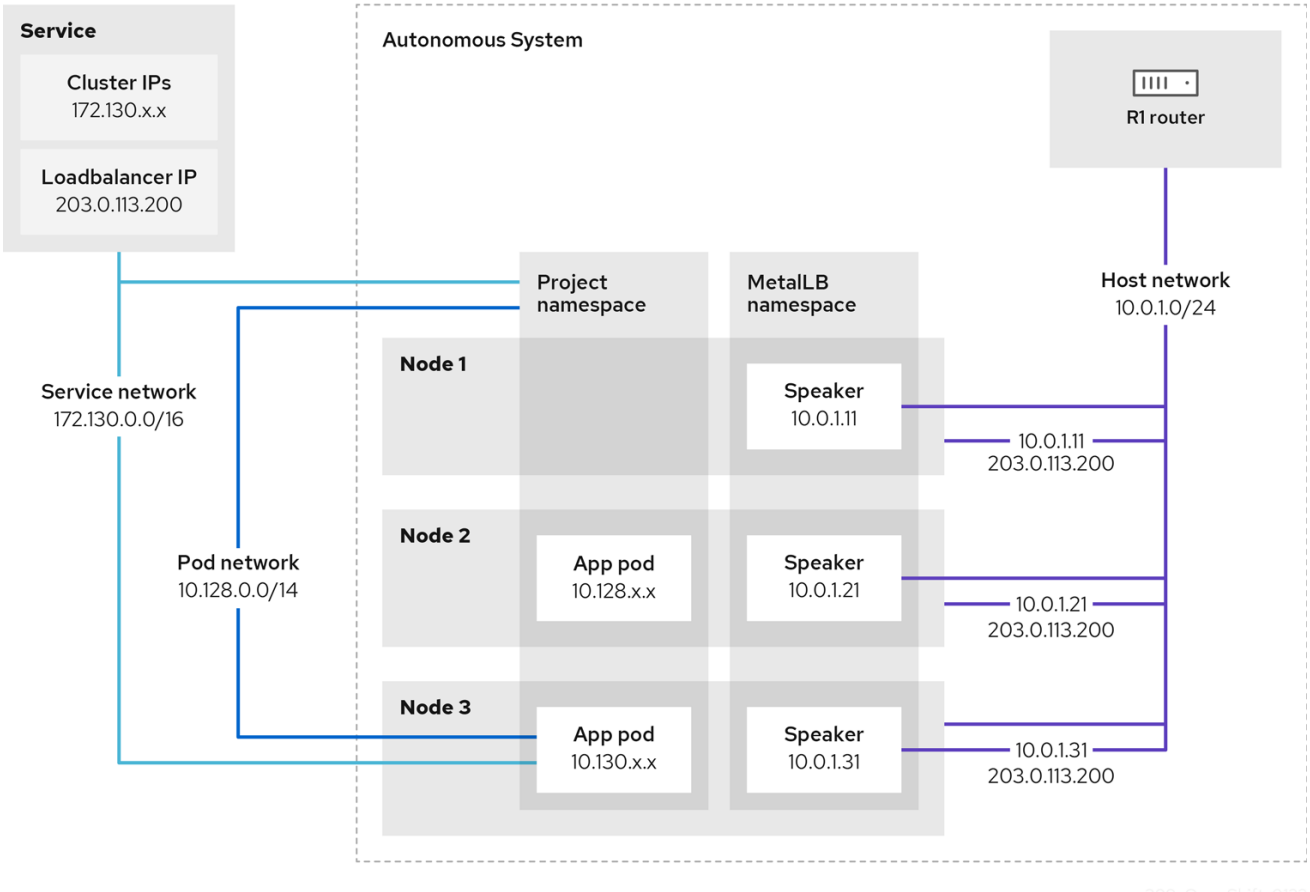
址。也可以通过添加可选 BGP 对等列表来广告来自给定池的 IP 地址到特定的对等池。BGP 对等点是配置为使用 BGP 协议的网络路由器。当路由器收到负载均衡器 IP 地址的流量时，路由器会选择一个带有公告 IP 地址的 speaker pod 的节点。路由器将流量发送到该节点。在流量进入节点后，CNI 网络插件的服务代理会将流量分发到该服务的所有 pod。

与集群节点相同的第 2 层网络段中直接连接的路由器可以配置为 BGP 对等点。如果直接连接的路由器没有配置为 BGP peer，您需要配置网络，以便负载均衡器 IP 地址的数据包在 BGP 对等机和运行 speaker Pod 的集群节点之间路由。

每次路由器接收负载均衡器 IP 地址的新流量时，它会创建一个新的与节点的连接。每个路由器制造商都有一个特定于实施的算法，用于选择要启动连接的节点。但是，算法通常设计为在可用节点之间分发流量，以平衡网络负载。

如果节点不可用，路由器会与具有 speaker pod 的另一个节点发起一个新的连接，以公告负载均衡器 IP 地址。

图 34.1. BGP 模式的 MetalLB 拓扑图



上图显示了与 MetalLB 相关的以下概念：

- 应用通过 172.130.0.0/16 子网上具有 IPv4 集群 IP 的服务进行访问。该 IP 地址可以从集群内部访问。该服务也有一个外部 IP 地址，MetalLB 分配到该服务 203.0.113.200。
- 节点 2 和 3 具有该应用的 pod。
- speaker 守护进程集在每个节点上运行一个 pod。MetalLB Operator 启动这些 pod。您可以配置 MetalLB 来指定运行 speaker pod 的节点。
- 每个 speaker pod 都是主机网络的 pod。容器集的 IP 地址与主机网络上节点的 IP 地址相同。

- 每个 speaker pod 启动一个 BGP 会话，其中包含所有 BGP 对等点，并将负载均衡器 IP 地址或聚合路由公告给 BGP 对等点。speaker pod 公告它们是 Autonomous System 65010 的一部分。图显示路由器 R1 作为同一自主系统内的 BGP peer。但是，您可以将 MetalLB 配置为与属于其他自主系统的同行启动 BGP 会话。
- 具有 speaker pod 的所有节点（公告负载均衡器 IP 地址）都可以接收该服务的流量。
  - 如果服务的外部流量策略设置为 cluster，则运行 speaker pod 的所有节点都会广告 203.0.113.200 负载平衡器 IP 地址，具有 speaker pod 的所有节点都可以接收该服务的流量。只有外部流量策略设为 cluster 时，主机前缀才会广告给路由器对等点。
  - 如果服务的外部流量策略设置为 local，则运行 speaker Pod 的所有节点都会运行，并且至少有一个运行的服务端点可能会广告 203.0.113.200 负载均衡器 IP 地址。只有这些节点才能接收该服务的流量。在上图中，节点 2 和 3 将公告 203.0.113.200。
- 您可以在添加 BGP peer 自定义资源时指定节点选择器，将 MetalLB 配置为通过指定带有特定 BGP peer 的节点选择器来控制哪些 speaker pod 启动 BGP 对等点。
- 任何配置为使用 BGP 的路由器（如 R1）都可以设置为 BGP 同级服务器。
- 客户端流量路由到主机网络上的其中一个节点。在流量进入节点后，服务代理会根据您为服务设置的外部流量策略，将流量发送到同一节点上的应用 pod 或其他节点。
- 如果节点不可用，路由器检测到失败，并启动与另一节点的新连接。您可以将 MetalLB 配置为将双向转发检测(BFD)配置集用于 BGP 对等点。BFD 提供更快的链路失败检测，以便路由器可以比没有 BFD 的情况下启动新连接。

34.1.7. 限制和限制

34.1.7.1. MetalLB 的基础架构注意事项

MetalLB 主要用于内部的裸机安装，因为这些安装不包含原生负载平衡器功能。除了裸机安装外，在有些基础架构上安装 OpenShift Container Platform 可能不包括原生负载均衡器功能。例如，以下基础架构可从添加 MetalLB Operator 中受益：

- 裸机
- VMware vSphere
- IBM Z® 和 IBM® LinuxONE
- IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM
- IBM Power®

OpenShift SDN 和 OVN-Kubernetes 网络供应商支持 MetalLB 和 MetalLB。

34.1.7.2. 第 2 层模式的限制

34.1.7.2.1. 单节点瓶颈

MetalLB 通过单一节点路由服务的所有流量，该节点可能会成为瓶颈并限制性能。

第 2 层模式将服务的入口带宽限制为单个节点的带宽。这是使用 ARP 和 NDP 定向流量的一个根本限制。

34.1.7.2.2. 延迟故障转移性能

节点之间的故障转移取决于客户端的合作。发生故障转移时，MetalLB 发送粒度 ARP 数据包来通知客户端与服务 IP 关联的 MAC 地址已更改。

大多数客户端操作系统正确处理细粒度 ARP 数据包，并及时更新其邻居缓存。当客户端快速更新其缓存时，故障转移将在几秒钟内完成。客户端通常在 10 秒内故障转移到新节点。但是，一些客户端操作系统或者根本不处理饱和的 ARP 数据包，或者存在延迟缓存更新的过时实施。

Windows、macOS 和 Linux 等常见操作系统的最新版本正确实现了第 2 层故障转移。除了较旧和不太常见的客户端操作系统外，预计不会出现故障转移较慢的问题。

为最大程度减轻计划内故障转移对过时客户端的影响，在颠倒领导地位后让旧节点保持运行几分钟。旧节点可以继续转发过期客户端的流量，直到其缓存刷新。

在计划外故障转移期间，服务 IP 无法访问，直到过期的客户端刷新其缓存条目为止。

34.1.7.2.3. 额外网络和 MetalLB 无法使用相同的网络

将相同的 VLAN 用于 MetalLB 和源 pod 上设置的额外网络接口可能会导致连接失败。当 MetalLB IP 和源 pod 驻留在同一节点上时，会出现这种情况。

为了避免连接失败，请将 MetalLB IP 放在源 pod 所在的不同子网中。此配置可确保来自源 pod 的流量将采用默认网关。因此，流量可以使用 OVN 覆盖网络有效地到达其目的地，确保连接功能如预期一样。

34.1.7.3. BGP 模式限制

34.1.7.3.1. 节点故障可能会破坏所有活跃的连接

MetalLB 共享一个限制，这是基于 BGP 的负载平衡。当 BGP 会话终止时，如节点失败或者 **speaker** pod 重启时，会话终止可能会导致重置所有活跃的连接。最终用户可以通过 **peer** 消息完成连接重置。

所终止的 BGP 会话的结果是特定于路由器制造商的实现。但是，您可以预测 **speaker** pod 数量的变化会影响 BGP 会话的数量，并且与 BGP 对等点的活动连接将中断。

为了避免或降低服务中断的可能性，您可以在添加 BGP 对等点时指定节点选择器。通过限制启动 BGP 会话的节点数量，没有 BGP 会话的节点出现错误不会影响到该服务的连接。

34.1.7.3.2. 只支持单个 ASN 和单个路由器 ID

当您添加 BGP peer 自定义资源时，您可以指定 **spec.myASN** 字段来识别 MetalLB 所属的 Autonomous System Number(ASN)。OpenShift Container Platform 使用带有 MetalLB 的 BGP 实施，它要求 MetalLB 属于单个 ASN。如果您试图添加 BGP peer 并为 **spec.myASN** 指定与现有的 BGP peer 自定义资源不同的值，您会收到一个错误。

同样，当您添加 BGP peer 自定义资源时，**spec.routerID** 字段是可选的。如果为此字段指定一个值，您必须为要添加的所有其他 BGP peer 自定义资源指定相同的值。

支持单个 ASN 和单个路由器 ID 的限制与支持的 MetalLB 实施不同。

34.1.8. 其他资源

- 比较：对外部 IP 地址进行容错访问
- 删除 IP 故障切换

- MetalLB 的部署规格

34.2. 安装 METALLB OPERATOR

作为集群管理员，您可以添加 MetalB Operator，以便 Operator 可以管理集群中的 MetalLB 实例的生命周期。

MetalLB 和 IP 故障转移不兼容。如果您为集群配置了 IP 故障切换，请在安装 Operator 前执行删除 IP 故障切换的步骤。

34.2.1. 使用 Web 控制台从 OperatorHub 安装 MetalLB Operator

作为集群管理员，您可以使用 OpenShift Container Platform Web 控制台安装 MetalLB Operator。

先决条件

- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 在 Filter by keyword 框中输入关键字，或滚动以查找您想要的 Operator。例如，键入 **metallb** 来查找 MetalLB Operator。
   您还可以根据基础架构功能过滤选项。例如，如果您希望 Operator 在断开连接的环境中工作，请选择 Disconnected。
3. 在 Install Operator 页面中，接受默认值并点 Install。

验证

1. 确认安装成功：
   a. 导航到 Operators → Installed Operators 页面。
   b. 检查 Operator 是否安装在 **openshift-operators** 命名空间中，其状态是否为 **Succeeded**。
2. 如果 Operator 没有成功安装，请检查 Operator 的状态并查看日志：
   a. 导航到 Operators → Installed Operators 页面，并检查 **Status** 列中是否有任何错误或故障。
   b. 导航到 Workloads → Pods 页面，并检查 **openshift-operators** 项目中报告问题的 pod 的日志。

34.2.2. 使用 CLI 从 OperatorHub 安装

您可以使用 CLI 从 OperatorHub 安装 Operator，而不必使用 OpenShift Container Platform Web 控制台。您可以使用 OpenShift CLI(**oc**)安装 MetalLB Operator。

建议您在使用 **metallb-system** 命名空间中安装 Operator 的 CLI 时使用。

先决条件

- 在裸机硬件上安装的集群。

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 输入以下命令为 MetalLB Operator 创建命名空间：

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

2. 在命名空间中创建 Operator 组自定义资源(CR)：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
EOF
```

3. 确认 Operator 组已安装在命名空间中：

```
$ oc get operatorgroup -n metallb-system
```

输出示例

```
NAME              AGE
metallb-operator   14m
```

4. 创建一个 Subscription CR：

   a. 定义 Subscription CR 并保存 YAML 文件，如metallb-sub.yaml ：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metallb-operator-sub
  namespace: metallb-system
spec:
  channel: stable
  name: metallb-operator
  source: redhat-operators (1)
  sourceNamespace: openshift-marketplace
```

   (1) 您必须指定 redhat-operators 值。

   b. 要创建 Subscription CR，请运行以下命令：

```
$ oc create -f metallb-sub.yaml
```

5. 可选：要确保 BGP 和 BFD 指标出现在 Prometheus 中，您可以使用以下命令标记命名空间：

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

验证

验证步骤假定 metallb-system 命名空间中安装了 MetalLB Operator。

1. 确认安装计划位于命名空间中：

```
$ oc get installplan -n metallb-system
```

输出示例

```
NAME            CSV                                APPROVAL    APPROVED
install-wzg94   metallb-operator.4.16.0-nnnnnnnnnnnn   Automatic   true
```

注意

安装 Operator 可能需要几秒钟。

2. 要验证是否已安装 Operator，请输入以下命令：

```
$ oc get clusterserviceversion -n metallb-system \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

输出示例

```
Name                                Phase
metallb-operator.4.16.0-nnnnnnnnnnnn   Succeeded
```

34.2.3. 在集群中启动 MetalLB

安装 Operator 后，您需要配置 MetalLB 自定义资源的单一实例。配置自定义资源后，Operator 会在集群中启动 MetalLB。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。
- 安装 MetalLB Operator。

流程

此流程假设 MetalLB Operator 已安装在 metallb-system 命名空间中。如果使用 Web 控制台安装，请替换命名空间的 openshift-operators。

1. 创建 MetalLB 自定义资源的单一实例：

```
$ cat << EOF | oc apply -f -
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
EOF
```

验证

确认 MetalLB 控制器的部署和 MetalLB speaker 的守护进程集正在运行。

1. 验证控制器的部署是否正在运行：

```
$ oc get deployment -n metallb-system controller
```

输出示例

```
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
controller   1/1     1            1           11m
```

2. 验证发言人的守护进程集是否正在运行：

```
$ oc get daemonset -n metallb-system speaker
```

输出示例

```
NAME     DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR          AGE
speaker   6       6       6     6            6           kubernetes.io/os=linux   18m
```

示例输出显示 6 个 speaker Pod。集群中的 speaker pod 数量可能与示例输出不同。确保输出指示集群中每个节点有一个容器集。

34.2.4. MetalLB 的部署规格

当使用 MetalLB 自定义资源启动 MetalLB 实例时，您可以在 MetalLB 自定义资源中配置部署规格，以管理 controller 或 speaker pod 如何在集群中部署并运行。使用这些部署规格来管理以下任务：

- 为 MetalLB pod 部署选择节点。
- 使用 pod 优先级和 pod 关联性来管理调度。
- 为 MetalLB pod 分配 CPU 限值。
- 为 MetalLB pod 分配容器 RuntimeClass。
- 为 MetalLB pod 分配元数据。

34.2.4.1. 将 speaker pod 限制到特定的节点

默认情况下，当使用 MetalLB Operator 启动 MetalLB 时，Operator 会在集群中的每个节点上启动 speaker pod 的实例。只有具有 speaker pod 的节点可以公告负载均衡器 IP 地址。您可以使用节点选择器配置 MetalLB 自定义资源，以指定运行 speaker pod 的节点。

将 speaker Pod 限制到特定的节点的最常见原因是，确保只有具有特定网络上网络接口的节点公告负载均衡器 IP 地址。只有具有正在运行的 speaker pod 的节点才会公告为负载均衡器 IP 地址的目的地。

如果将 speaker 的 pod 限制到特定的节点，并为服务的外部流量策略指定 local，则必须确保该服务的应用程序 pod 部署到同一节点上。

将 speaker pod 限制为 worker 节点的配置示例

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector: 1
    node-role.kubernetes.io/worker: ""
  speakerTolerations: 2
  - key: "Example"
    operator: "Exists"
    effect: "NoExecute"
```

1 示例配置指定将 speaker pod 分配给 worker 节点，但您可以指定分配给节点或任何有效的节点选择器的标签。

2 在本示例配置中，附加此容限的 pod 可以容忍与 key 值和 effect 值匹配的任何污点，并使用 operator 容许值。

使用 spec.nodeSelector 字段应用清单后，您可以检查 Operator 使用 oc get daemonset -n metallb-system speaker 命令部署的 pod 数量。同样，您可以使用 oc get nodes -l node-role.kubernetes.io/worker= 等命令显示与标签匹配的节点。

您可以选择允许节点使用关联性规则控制哪些 speaker pod 应该或不应该调度到节点上。您还可以通过应用容限列表来限制这些 pod。如需有关关联性规则、污点和容限的更多信息，请参阅其他资源。

34.2.4.2. 在 MetalLB 部署中配置容器运行时类

您可以通过配置 MetalLB 自定义资源，选择将容器运行时类分配给 controller 和 speaker pod。例如，对于 Windows 工作负载，您可以将 Windows 运行时类分配给 pod，它将这个运行时类用于 pod 中所有容器。

先决条件

- 您以具有 cluster-admin 权限的用户身份登录。
- 已安装 MetalLB Operator。

流程

1. 创建 RuntimeClass 自定义资源，如 myRuntimeClass.yaml，以定义您的运行时类：

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
```

```
  name: myclass
handler: myconfiguration
```

2. 应用 RuntimeClass 自定义资源配置：

```
$ oc apply -f myRuntimeClass.yaml
```

3. 创建 MetalLB 自定义资源，如 MetalLBRuntime.yaml，以指定 runtimeClassName 值：

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    runtimeClassName: myclass
    annotations: 1
      controller: demo
  speakerConfig:
    runtimeClassName: myclass
    annotations:
      speaker: demo
```

1 本例使用 注解 来添加元数据，如构建发行信息或 GitHub 拉取请求信息。您可以使用标签中不允许的字符填充注解。但是，您无法使用注解来标识或选择对象。

4. 应用 MetalLB 自定义资源配置：

```
$ oc apply -f MetalLBRuntime.yaml
```

验证

- 要查看 pod 的容器运行时，请运行以下命令：

```
$ oc get pod -o custom-
columns=NAME:metadata.name,STATUS:.status.phase,RUNTIME_CLASS:.spec.runti
meClassName
```

34.2.4.3. 在 MetalLB 部署中配置 pod 优先级和 pod 关联性

您可以通过配置 MetalLB 自定义资源，选择将 pod 优先级和 pod 关联性规则分配给controller 和 speaker pod。pod 优先级指示节点上 pod 的相对重要性，并根据这个优先级调度 pod。在controller 或 speaker pod 上设置高优先级，以确保在节点上的其他 pod 上调度优先级。

Pod 关联性管理 pod 间的关系。将 pod 关联性分配给 controller 或 speaker pod，以控制调度程序将 pod 放置到 pod 关系的节点上。例如，您可以使用 pod 关联性规则来确保某些 pod 位于同一节点或节点上，这有助于提高网络通信并减少这些组件之间的延迟。

先决条件

- 您以具有 cluster-admin 权限的用户身份登录。

- 已安装 MetalLB Operator。
- 已在集群中启动 MetalLB Operator。

流程

1. 创建 PriorityClass 自定义资源，如 myPriorityClass.yaml，以配置优先级级别。这个示例定义了名为 high-priority 的 PriorityClass，值设为 1000000。与具有较低优先级类的 pod 相比，在调度过程中分配此优先级类的 Pod 被视为优先级更高：

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
```

2. 应用 PriorityClass 自定义资源配置：

```
$ oc apply -f myPriorityClass.yaml
```

3. 创建 MetalLB 自定义资源，如 MetalLBPodConfig.yaml，以指定 priorityClassName 和 podAffinity 值：

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    priorityClassName: high-priority (1)
    affinity:
      podAffinity: (2)
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: metallb
          topologyKey: kubernetes.io/hostname
  speakerConfig:
    priorityClassName: high-priority
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: metallb
          topologyKey: kubernetes.io/hostname
```

(1) 指定 MetalLB 控制器 pod 的优先级类。在这种情况下，它被设置为 high-priority。

(2) 指定您要配置 pod 关联性规则。这些规则指定如何调度 pod 与其他 pod 或节点。此配置指示调度程序将具有标签 app: metallb 的 pod 调度到共享同一主机名的节点。这有助于在同一节点上并置与 MetalLB 相关的 pod，可能会优化这些 pod 之间的网络通信、延迟和资源使用情况。

4. 应用 MetalLB 自定义资源配置：

```
$ oc apply -f MetalLBPodConfig.yaml
```

验证

- 要在 metallb-system 命名空间中查看分配给 pod 的优先级类，请运行以下命令：

```
$ oc get pods -n metallb-system -o custom-
columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName
```

输出示例

```
NAME                                  PRIORITY
controller-584f5c8cd8-5zbvg           high-priority
metallb-operator-controller-manager-9c8d9985-szkqg   <none>
metallb-operator-webhook-server-c895594d4-shjgx      <none>
speaker-dddf7                         high-priority
```

- 要验证调度程序是否根据 pod 关联性规则放置 pod，请运行以下命令来查看 pod 的节点或节点的元数据：

```
$ oc get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name -n
metallb-system
```

34.2.4.4. 在 MetalLB 部署中配置 pod CPU 限制

您可以通过配置 MetalLB 自定义资源（可选）将 pod CPU 限值分配给controller 和 speaker pod。为 controller 或 speaker pod 定义 CPU 限制可帮助您管理节点上的计算资源。这样可确保节点上的所有 pod 具有必要的计算资源来管理工作负载和集群内务。

先决条件

- 您以具有 cluster-admin 权限的用户身份登录。
- 已安装 MetalLB Operator。

流程

1. 创建 MetalLB 自定义资源文件，如 CPULimits.yaml，以指定 controller 和 speaker pod 的cpu 值：

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    resources:
      limits:
        cpu: "200m"
```

```
    speakerConfig:
      resources:
        limits:
          cpu: "300m"
```

2. 应用 MetalLB 自定义资源配置：

```
$ oc apply -f CPULimits.yaml
```

验证

- 要查看 pod 的计算资源，请运行以下命令，将 <pod_name> 替换为您的目标 pod：

```
$ oc describe pod <pod_name>
```

34.2.5. 其他资源

- 使用节点选择器将 pod 放置到特定节点
- 了解污点和容限
- 了解 pod 优先级
- 了解 pod 关联性

34.2.6. 后续步骤

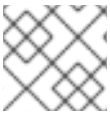
- 配置 MetalLB 地址池

34.3. 升级 METALLB

请注意，如果您当前运行的是版本 4.10，或 MetalLB Operator 的早期版本，对 4.10 之后的任何版本的自动更新都无法正常工作。从 4.11 或更高版本的 MetalLB Operator 版本升级到更新的版本。例如，从 4.12 升级到 4.13 版本将会平稳进行。

从 4.10 及更早版本的 MetalLB Operator 的升级步骤概述如下：

1. 删除安装的 MetalLB Operator 版本，如 4.10。确保没有删除命名空间和 metallb 自定义资源。
2. 使用 CLI，在安装之前版本的 MetalLB Operator 的同一命名空间中安装 MetalLB Operator 4.16。



注意

此流程不适用于 MetalLB Operator 的自动 z-stream 更新，这遵循标准的方法。

有关从 4.10 及更早版本升级 MetalLB Operator 的详细信息，请参阅以下指导。作为集群管理员，使用 OpenShift CLI (oc) 或 Web 控制台删除 MetalLB Operator 来启动升级过程。

34.3.1. 使用 Web 控制台从集群中删除 MetalLB Operator

集群管理员可以使用 Web 控制台从所选命名空间中删除已安装的 Operator。

先决条件

- **使用具有 cluster-admin 权限的账户访问 OpenShift Container Platform 集群 Web 控制台。**

流程

1. **导航到 Operators → Installed Operators 页面。**
2. **搜索 MetalLB Operator。然后点它。**
3. **在 Operator Details 页面右侧，从 Actions 下拉菜单中选择 Uninstall Operator。**
   此时会显示 Uninstall Operator? 对话框。
4. **选择 Uninstall 来删除 Operator、Operator 部署和 pod。按照此操作，Operator 将停止运行，不再接收更新。**

注意

此操作不会删除 Operator 管理的资源，包括自定义资源定义 (CRD) 和自定义资源 (CR)。Web 控制台和继续运行的集群资源启用的仪表板和导航项可能需要手动清理。要在卸载 Operator 后删除这些，您可能需要手动删除 Operator CRD。

34.3.2. 使用 CLI 从集群中删除 MetalLB Operator

集群管理员可以使用 CLI 从所选命名空间中删除已安装的 Operator。

先决条件

- **使用具有 cluster-admin 权限的账户访问 OpenShift Container Platform 集群。**
- **已在工作站上安装 oc 命令。**

流程

1. **在 currentCSV 字段中检查订阅的 MetalLB Operator 的当前版本：**

```
$ oc get subscription metallb-operator -n metallb-system -o yaml | grep currentCSV
```

输出示例

```
  currentCSV: metallb-operator.4.10.0-202207051316
```

2. **删除订阅：**

```
$ oc delete subscription metallb-operator -n metallb-system
```

输出示例

```
subscription.operators.coreos.com "metallb-operator" deleted
```

3. **使用上一步中的 currentCSV 值来删除目标命名空间中相应 Operator 的 CSV：**

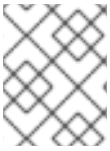
```
$ oc delete clusterserviceversion metallb-operator.4.10.0-202207051316 -n metallb-system
```

输出示例

```
clusterserviceversion.operators.coreos.com "metallb-operator.4.10.0-202207051316" deleted
```

34.3.3. 编辑 MetalLB Operator Operator 组

当从任何 MetalLB Operator 版本升级到并包括 4.10 升级到 4.11 及之后的版本时，从 Operator 组自定义资源(CR)中删除 **spec.targetNamespaces**。无论是否使用 Web 控制台或 CLI 来删除 MetalLB Operator，都必须删除 spec。



注意

MetalLB Operator 版本 4.11 或更高版本只支持 **AllNamespaces** 安装模式，而 4.10 或更早的版本支持 **OwnNamespace** 或 **SingleNamespace** 模式。

先决条件

- 您可以使用 **cluster-admin** 权限访问 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 运行以下命令，列出 **metallb-system** 命名空间中的 Operator 组：

```
$ oc get operatorgroup -n metallb-system
```

输出示例

```
NAME                 AGE
metallb-system-7jc66   85m
```

2. 运行以下命令，验证与 **metallb-system** 命名空间关联的 Operator 组 CR 中是否存在 **spec.targetNamespaces** ：

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

输出示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 1
  name: metallb-system-7jc66
```

```
  namespace: metallb-system
  resourceVersion: "25027"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  targetNamespaces:
  - metallb-system
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T09:42:49Z"
  namespaces:
  - metallb-system
```

3. 运行以下命令，编辑 Operator 组并删除 spec 部分下的 targetNamespaces 和 metallb-system：

```
$ oc edit n metallb-system
```

输出示例

```
operatorgroup.operators.coreos.com/metallb-system-7jc66 edited
```

4. 运行以下命令，验证 spec.targetNamespaces 已从与 metallb-system 命名空间关联的 Operator 组自定义资源中删除：

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

输出示例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 2
  name: metallb-system-7jc66
  namespace: metallb-system
  resourceVersion: "61658"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T14:31:30Z"
  namespaces:
  - ""
```

34.3.4. 升级 MetalLB Operator

先决条件

- 使用具有 cluster-admin 角色的用户访问集群。

流程

1. 验证 metallb-system 命名空间仍然存在：

```
$ oc get namespaces | grep metallb-system
```

输出示例

```
metallb-system                         Active   31m
```

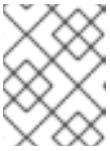
2. 验证 metallb 自定义资源仍然存在：

```
$ oc get metallb -n metallb-system
```

输出示例

```
NAME     AGE
metallb   33m
```

3. 按照"使用 CLI 安装 OperatorHub"中的指导来安装 MetalLB Operator 的最新 4.16 版本。



注意

安装最新的 MetalLB Operator 版本 4.16 时，您必须将 Operator 安装到之前安装的同一命名空间。

4. 验证 Operator 的升级版本现在是 4.16 版本。

```
$ oc get csv -n metallb-system
```

输出示例

```
NAME                                    DISPLAY         VERSION              REPLACES   PHASE
metallb-operator.4.16.0-202207051316   MetalLB Operator   4.16.0-202207051316
Succeeded
```

34.3.5. 其他资源

- 从集群中删除 Operator
- 安装 MetalLB Operator

34.4. 配置 METALLB 地址池

作为集群管理员，您可以添加、修改和删除地址池。MetalLB Operator 使用地址池自定义资源来设置 MetalLB 可分配给服务的 IP 地址。示例中使用的命名空间假定命名空间是 metallb-system。

34.4.1. 关于 IPAddressPool 自定义资源

下表中描述了 IPAddressPool 自定义资源的字段。

表 34.1. MetalLB IPAddressPool 池自定义资源

字段	类型	描述
metadata.name	**string**	指定地址池的名称。添加服务时，您可以在 **metallb.universe.tf/address-pool** 注解中指定这个池名称，以从特定池中选择 IP 地址。整个文档中都使用名称 **doc-example**、**silver** 和 **gold**。
metadata.namespace	**string**	指定地址池的命名空间。指定 MetalLB Operator 使用的同一命名空间。
metadata.label	**string**	可选：指定分配给 **IPAddressPool** 的键值对。这可通过 **BGPAdvertisement** and **L2Advertisement** CRD 中的 **ipAddressPoolSelectors** 进行引用，以将 **IPAddressPool** 与广告关联
spec.addresses	**string**	指定分配给服务的 MetalLB Operator 的 IP 地址列表。您可以在一个池中指定多个范围 ; 它们将共享相同的设置。以 CIDR 表示法指定每个范围，或者指定为以连字符隔开的起始和结束 IP 地址。
spec.autoAssign	**布尔值**	可选：指定 MetalLB 是否从这个池自动分配 IP 地址。如果要使用 **metallb.universe.tf/address-pool** 注解从这个池中明确请求 IP 地址，请指定 **false**。默认值为 **true**。
spec.avoidBuggyIPs	**布尔值**	可选：当启用时，IP 地址以 .0 和 .255 结尾时，不会从池中分配。默认值为 **false**。某些较旧的消费者网络设备错误地阻止以 .0 和 .255 结尾的 IP 地址。

您可以通过配置 spec.serviceAllocation 规格，将 IP 地址从 IPAddressPool 分配给服务和命名空间。

表 34.2. MetalLB IPAddressPool 自定义资源 spec.serviceAllocation 子字段

字段	类型	描述
priority	**int**	可选：当多个 IP 地址池与服务或命名空间匹配时，定义 IP 地址池之间的优先级。较低数字表示优先级更高。
命名空间	**数组（字符串）**	可选：指定您可以分配给 IP 地址池 IP 地址的命名空间列表。
namespaceSelectors	**数组 (LabelSelector)**	可选：指定您可以使用列表格式的标签选择器从 IP 地址池分配给 IP 地址的命名空间标签。
serviceSelectors	**数组 (LabelSelector)**	可选：指定您可以使用列表格式的标签选择器从地址池中分配给 IP 地址的服务标签。

34.4.2. 配置地址池

作为集群管理员，您可以在集群中添加地址池来控制 MetalLB 可分配给负载均衡器服务的 IP 地址。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example
  labels: 1
    zone: east
spec:
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75
```

1 分配给 IPAddressPool 的该标签可通过 BGPAdvertisement CRD 中的 ipAddressPoolSelectors 来引用，以将 IPAddressPool 与广告关联。

2. 应用 IP 地址池的配置：

```
$ oc apply -f ipaddresspool.yaml
```

验证

- 查看地址池：

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

输出示例

```
Name:         doc-example
Namespace:    metallb-system
Labels:       zone=east
Annotations:  <none>
API Version:  metallb.io/v1beta1
Kind:         IPAddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign:  true
Events:         <none>
```

确认输出中显示了地址池名称，如 doc-example，并且 IP 地址范围显示在输出中。

34.4.3. 地址池配置示例

34.4.3.1. 示例：IPv4 和 CIDR 范围

您可以使用 CIDR 表示法指定 IP 地址范围。您可以将 CIDR 表示法与使用连字符分隔下限和上限的表示法合并。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  addresses:
  - 192.168.100.0/24
  - 192.168.200.0/24
  - 192.168.255.1-192.168.255.5
```

34.4.3.2. 示例：保留 IP 地址

您可以将 autoAssign 字段设置为 false，以防止 MetalLB 自动从池中分配 IP 地址。添加服务时，您可以从池中请求特定的 IP 地址，或者在注解中指定池名称从池中请求任何 IP 地址。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  autoAssign: false
```

34.4.3.3. 示例：IPv4 和 IPv6 地址

您可以添加使用 IPv4 和 IPv6 的地址池。您可以像几个 IPv4 示例一样在 地址 列表中指定多个范围。

无论服务被分配一个 IPv4 地址、一个 IPv6 地址，还是由您添加该服务来确定。spec.ipFamilies 和 spec.ipFamilyPolicy 字段控制 IP 地址如何分配给该服务。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  - 2002:2:2::1-2002:2:2::100
```

34.4.3.4. 示例：将 IP 地址池分配给服务或命名空间

您可以将 IP 地址从 IPAddressPool 分配给您指定的服务和命名空间。

如果您将服务或命名空间分配给多个 IP 地址池，MetalLB 将使用较高优先级 IP 地址池中的可用 IP 地址。如果分配的 IP 地址池没有 IP 地址可用，MetalLB 将使用较低优先级或没有优先级的 IP 地址池的可用 IP 地址。

注意

对于 namespaceSelectors 和 serviceSelectors 规格，您可以使用 matchLabels 标签选择器、matchExpressions 标签选择器或两者。本例展示了每个规格的一个标签选择器。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-service-allocation
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50 (1)
    namespaces: (2)
      - namespace-a
      - namespace-b
    namespaceSelectors: (3)
      - matchLabels:
          zone: east
    serviceSelectors: (4)
      - matchExpressions:
        - key: security
          operator: In
          values:
          - S1
```

(1) 为地址池分配优先级。较低数字表示优先级更高。

(2) 以列表格式为 IP 地址池分配一个或多个命名空间。

(3) 使用列表格式的标签选择器为 IP 地址池分配一个或多个命名空间标签。

(4) 使用列表格式的标签选择器，为 IP 地址池分配一个或多个服务标签。

34.4.4. 其他资源

- 使用 L2 广告和标签配置 MetalLB。

34.4.5. 后续步骤

- 对于 BGP 模式，请参阅 配置 MetalLB BGP peer。
- 配置服务以使用 MetalLB。

34.5. 关于 IP 地址池的广告

您可以配置 MetalLB，以便使用第 2 层协议、BGP 协议或两者来广告 IP 地址。通过第 2 层，MetalLB 提供了容错的外部 IP 地址。使用 BGP，MetalLB 为外部 IP 地址和负载均衡提供容错功能。

MetalLB 支持将 L2 和 BGP 用于同一组 IP 地址。

MetalLB 提供了将地址池分配给特定 BGP 对等对象到网络上节点子集的灵活性。这可以实现更复杂的配置，例如促进节点隔离或网络分段。

34.5.1. 关于 BGPAdvertisement 自定义资源

BGPAdvertise 对象的字段在下表中定义：

表 34.3. BGPAdvertise 配置

字段	类型	描述
metadata.name	**string**	指定 BGP 广告的名称。
metadata.namespace	**string**	指定 BGP 广告的命名空间。指定 MetalLB Operator 使用的同一命名空间。
spec.aggregationLength	**整数**	可选：指定 32 位 CIDR 掩码中包含的位数。为了聚合发言人向 BGP 对等者公告的路由，掩码将应用于多个服务 IP 地址的路由，speaker 会公告聚合的路由。例如，聚合长度为 **24**，speaker 可以聚合多个 **10.0.1.x/32** 服务 IP 地址并公告一个 **10.0.1.0/24** 路由。
spec.aggregationLengthV6	**整数**	可选：指定 128 位 CIDR 掩码中包含的位数。例如，在聚合长度为 **124** 时，speaker 可以聚合几个 **fc00:f853:0ccd:e799::x/128** 服务 IP 地址，并公告一个 **fc00:f853:0ccd:e799::0/124** 路由。
spec.communities	**string**	可选：指定一个或多个 BGP 社区。每个社区都被指定为两个 16 位值，用冒号字符分隔。知名的社区必须指定为 16 位值： • **NO_EXPORT**: **65535:65281** • **NO_ADVERTISE**: **65535:65282** • **NO_EXPORT_SUBCONFED**: **65535:65283**  **注意** 您还可以使用与字符串一起创建的社区对象。
spec.localPref	**整数**	可选：指定这个广播的本地首选项。此 BGP 属性适用于 Autonomous System 中的 BGP 会话。
spec.ipAddressPools	**string**	可选：用于使用这个广告进行广告的 **IPAddressPools** 列表，按名称选择。

字段	类型	描述
spec.ipAddressPoolSelectors	string	可选：使用这个广告进行广告的 IPAddressPools 的选择器。这可用于根据分配给 IPAddressPool 而非名称本身的标签将 IPAddressPool 与公告关联。如果这个或列表没有选择 IPAddressPool，则公告会应用到所有 IPAddressPools。
spec.nodeSelectors	string	可选：NodeSelectors 允许限制节点作为负载均衡器 IP 的下一个跃点。如果为空，所有节点都会作为下一个跃点进行宣布。
spec.peers	string	可选： Peers 限制 BGP 对等点来公告所选池的 IP。如果为空，负载均衡器 IP 会声明所有配置了 BGP 对等点。

34.5.2. 使用 BGP 公告和基本用例配置 MetalLB

按如下所示配置 MetalLB，使对等 BGP 路由器为每个 MetalLB 分配为服务的负载均衡器 IP 地址接收一个 203.0.113.200/32 路由和一个 fc00:f853:ccd:e799::1/128 路由。因为没有指定 localPref 和 community 字段，所以路由会公告，并将 localPref 设置为 0，且没有 BGP 社区。

34.5.2.1. 示例：使用 BGP 传输基本地址池配置

按如下所示配置 MetalLB，以便使用 BGP 协议公告 IPAddressPool。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**

流程

1. **创建 IP 地址池。**
   a. **创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：**

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-basic
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
```

   b. **应用 IP 地址池的配置：**

```
$ oc apply -f ipaddresspool.yaml
```

2. **创建 BGP 公告。**

a. 创建一个文件，如 bgpadvertisement.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-basic
  namespace: metallb-system
spec:
  ipAddressPools:
  - doc-example-bgp-basic
```

b. 应用配置：

```
$ oc apply -f bgpadvertisement.yaml
```

34.5.3. 使用 BGP 广告和高级用例配置 MetalLB

按如下所示配置 MetalLB，使得 MetalLB 分配给负载均衡器的 IP 地址范围是 203.0.113.200 到 203.0.113.203，以及 fc00:f853:ccd:e799::0 到 fc00:f853:ccd:e799::f。

为了说明两个 BGP 公告，在 MetalLB 分配 IP 地址 203.0.113.200 时，请考虑实例。以该 IP 地址为例，发言人向 BGP 对等点公告两个路由：

- 203.0.113.200/32，localPref 设置为 100，社区设置为 NO_ADVERTISE 社区的数字值。此规范指示它们可以使用此路由的对等路由器，但它们不应将有关此路由的信息传播到 BGP 对等点。
- 203.0.113.200/30 将 MetalLB 分配的负载均衡器 IP 地址聚合到一个路由中。MetalLB 公告到 BGP 对等点的聚合路由，并将 community 属性设置为 8000:800。BGP 同行将 203.0.113.200/30 个路由传播到其他 BGP 同级服务器。当流量通过发言人路由到节点时，使用 203.0.113.200/32 路由将流量转发到集群以及与该服务关联的 pod。

当添加更多服务和 MetalLB 从池中分配更多负载均衡器 IP 地址时，对等路由器收到一个本地路由，203.0.113.20x/32，以及 203.0.113.200/30 聚合路由。您添加的每个服务都会生成 /30 路由，但 MetalLB 会将路由重复数据删除到一个 BGP 公告，然后再与对等路由器通信。

34.5.3.1. 示例：使用 BGP 传输高级地址池配置

按如下所示配置 MetalLB，以便使用 BGP 协议公告 IPAddressPool。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建 IP 地址池。

   a. 创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
```

```
  name: doc-example-bgp-adv
  labels:
    zone: east
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
  autoAssign: false
```

b. 应用 IP 地址池的配置：

```
$ oc apply -f ipaddresspool.yaml
```

2. 创建 BGP 公告。

a. 创建一个文件，如 bgpadvertisement1.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 65535:65282
  aggregationLength: 32
  localPref: 100
```

b. 应用配置：

```
$ oc apply -f bgpadvertisement1.yaml
```

c. 创建一个文件，如 bgpadvertisement2.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 8000:800
  aggregationLength: 30
  aggregationLengthV6: 124
```

d. 应用配置：

```
$ oc apply -f bgpadvertisement2.yaml
```

34.5.4. 从节点的子集公告 IP 地址池

要从 IP 地址池公告 IP 地址，请只使用特定的节点集合，使用 BGPAdvertisement 自定义资源中的 .spec.nodeSelector 规格。此规格将 IP 地址池与集群中的一组节点关联。如果您在集群中的不同子网上有节点，而您想要从特定子网的地址池中公告 IP 地址，例如仅面向公共的子网。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 使用自定义资源创建 IP 地址池：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
```

2. 通过在 BGPAdvertisement 自定义资源中定义 .spec.nodeSelector 值，控制 pool1 广告的 IP 地址来自集群中的哪些节点：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example
spec:
  ipAddressPools:
  - pool1
  nodeSelector:
  - matchLabels:
      kubernetes.io/hostname: NodeA
  - matchLabels:
      kubernetes.io/hostname: NodeB
```

在本例中，pool1 广告的 IP 地址仅来自 NodeA 和 NodeB。

34.5.5. 关于 L2Advertisement 自定义资源

l2Advertise 对象的字段在下表中定义：

表 34.4. L2 广告配置

字段	类型	描述
metadata.name	string	指定 L2 广告的名称。

字段	类型	描述
metadata.namespace	**string**	指定 L2 广告的命名空间。指定 MetalLB Operator 使用的同一命名空间。
spec.ipAddressPools	**string**	可选：用于使用这个广告进行广告的 **IPAddressPools** 列表，按名称选择。
spec.ipAddressPoolSelectors	**string**	可选：使用这个广告进行广告的 **IPAddressPools** 的选择器。这可用于根据分配给 **IPAddressPool** 而非名称本身的标签将 **IPAddressPool** 与公告关联。如果这个或列表没有选择 **IPAddressPool**，则公告会应用到所有 **IPAddressPools**。
spec.nodeSelectors	**string**	可选：**NodeSelectors** 将节点限制在负载均衡器 IP 的下一跃点中。如果为空，所有节点都会作为下一个跃点进行宣布。 **重要** 限制节点，因为下一跃点只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。 有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。
spec.interfaces	**string**	可选：用于声明负载均衡器 IP 的**接口**列表。

34.5.6. 使用 L2 广告配置 MetalLB

按如下所示配置 MetalLB，以便使用 L2 协议广告 IPAddressPool。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**

流程

1. **创建 IP 地址池。**
   a. **创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：**

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
```

```
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

b. 应用 IP 地址池的配置：

```
$ oc apply -f ipaddresspool.yaml
```

2. 创建 L2 广告。

a. 创建一个文件，如 l2advertisement.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
```

b. 应用配置：

```
$ oc apply -f l2advertisement.yaml
```

34.5.7. 使用 L2 广告和标签配置 MetalLB

BGPAdvertisement 和 L2Advertisement 中的 ipAddress Pools 字段用于根据分配给 IPAddressPool 的标签将 IPAddressPool 与广告相关联。

本例演示了如何配置 MetalLB，以便通过配置 ipAddressPoolSelectors 字段来广告使用 L2 协议的 IPAddressPools。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建 IP 地址池。

a. 创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2-label
  labels:
    zone: east
```

```
spec:
  addresses:
    - 172.31.249.87/32
```

b. 应用 IP 地址池的配置：

```
$ oc apply -f ipaddresspool.yaml
```

2. 使用 ipAddressPoolSelectors 创建 L2 广告广告 IP。

a. 创建一个文件，如 l2advertisement.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
        - key: zone
          operator: In
          values:
            - east
```

b. 应用配置：

```
$ oc apply -f l2advertisement.yaml
```

34.5.8. 为所选接口配置带有 L2 广告的 MetalLB

默认情况下，分配给该服务的 IP 地址池的 IP 地址将从所有网络接口公告。L2Advertisement 自定义资源定义中的 interfaces 字段用于限制公告 IP 地址池的网络接口。

本例演示了如何配置 MetalLB，以便仅从所有节点的 interfaces 字段中列出的网络接口公告 IP 地址池。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您以具有 cluster-admin 权限的用户身份登录。

流程

1. 创建 IP 地址池。

a. 创建一个文件，如 ipaddresspool.yaml，并输入类似以下示例的配置详情：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
```

```
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

b. 为 IP 地址池应用配置，如下例所示：

```
$ oc apply -f ipaddresspool.yaml
```

2. 使用 接口 选择器创建 L2 广告广告 IP。

a. 创建一个 YAML 文件，如 l2advertisement.yaml，并输入类似以下示例的配置详情：

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
   interfaces:
   - interfaceA
   - interfaceB
```

b. 为公告应用配置，如下例所示：

```
$ oc apply -f l2advertisement.yaml
```

重要

接口选择器不会影响 MetalLB 如何使用 L2 选择给定 IP 的节点。如果节点没有所选接口，则所选节点不会宣布该服务。

34.5.9. 其他资源

- 配置社区别名。

34.6. 配置 METALLB BGP PEER

作为集群管理员，您可以添加、修改和删除边框网关协议(BGP)对等点。MetalLB Operator 使用 BGP peer 自定义资源来标识 MetalLB speaker pod 联系的对等者启动 BGP 会话。对等点接收 MetalLB 分配给服务的负载均衡器 IP 地址的路由公告。

34.6.1. 关于 BGP peer 自定义资源

下表中描述了 BGP peer 自定义资源的字段。

表 34.5. MetalLB BGP peer 自定义资源

字段	类型	描述
metadata.name	string	指定 BGP peer 自定义资源的名称。

字段	类型	描述
metadata.namespace	**string**	指定 BGP peer 自定义资源的命名空间。
spec.myASN	**整数**	为 BGP 会话的本地末尾指定 Autonomous System 号。在您添加的所有 BGP peer 自定义资源中指定相同的值。范围是从 **0** 到 **4294967295**。
spec.peerASN	**整数**	为 BGP 会话的远程端指定 Autonomous System 号。范围是从 **0** 到 **4294967295**。
spec.peerAddress	**string**	指定建立 BGP 会话的对等点的 IP 地址。
spec.sourceAddress	**string**	可选：指定建立 BGP 会话时要使用的 IP 地址。该值必须是 IPv4 地址。
spec.peerPort	**整数**	可选：指定用来建立 BGP 会话的对等端口。范围为 **0** 到 **16384**。
spec.holdTime	**string**	可选：指定到 BGP 对等点的保留时间。最小值为 3 秒(**3s**)。常见的单位是秒和分钟，如 **3s**、**1m** 和 **5m30s**。要更快地检测路径失败，还要配置 BFD。
spec.keepaliveTime	**string**	可选：指定向 BGP 对等发送保留消息之间的最大间隔。如果指定此字段，还必须为 **holdTime** 字段指定一个值。指定的值必须小于 **holdTime** 字段的值。
spec.routerID	**string**	可选：指定要公告到 BGP peer 的路由器 ID。如果指定了此字段，则必须在添加的每个 BGP peer 自定义资源中指定相同的值。
spec.password	**string**	可选：指定 MD5 密码，以发送到执行 TCP MD5 经过身份验证的 BGP 会话的路由器的对等点。
spec.passwordSecret	**string**	可选：指定 BGP Peer 的身份验证 secret 的名称。secret 必须存在于 **metallb** 命名空间中，且类型为 basic-auth。
spec.bfdProfile	**string**	可选：指定 BFD 配置集的名称。
spec.nodeSelectors	**object[]**	可选：使用匹配表达式和匹配标签指定选择器，以控制哪些节点可以连接到 BGP 对等点。
spec.ebgpMultiHop	**布尔值**	可选：指定 BGP peer 是否有多个网络跃点。如果 BGP peer 没有直接连接到同一网络，则 speaker 无法建立 BGP 会话，除非此字段设置为 **true**。此字段适用于 *外部 BGP*。外部 BGP 是用来描述当 BGP 对等点属于不同的自治系统的术语。

注意

passwordSecret 字段与 password 字段相互排斥，包含对包含密码的 secret 的引用。设置这两个字段会导致解析失败。

34.6.2. 配置 BGP peer

作为集群管理员，您可以添加 BGP peer 自定义资源来与网络路由器交换路由信息，并为服务公告 IP 地址。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**
- **使用 BGP 公告配置 MetalLB。**

流程

1. **创建一个文件，如 bgppeer.yaml，其内容类似以下示例：**

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

2. **应用 BGP peer 的配置：**

```
$ oc apply -f bgppeer.yaml
```

34.6.3. 为给定地址池配置一组特定的 BGP 对等组

此流程演示了如何：

- **配置一组地址池（pool1 和 pool2）。**
- **配置一组 BGP 对等点（pe1 和 peer2）。**
- **配置 BGP 广告，将 pool1 分配给 peer1，将 pool2 分配给 peer2。**

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**

流程

1. 创建地址池 pool1。

   a. 创建一个文件，如 ipaddresspool1.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
```

   b. 为 IP 地址池 pool1 应用配置：

```
$ oc apply -f ipaddresspool1.yaml
```

2. 创建地址池 pool2。

   a. 创建一个文件，如 ipaddresspool2.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool2
spec:
  addresses:
    - 5.5.5.100-5.5.5.200
    - 2001:100:5::200-2001:100:5::400
```

   b. 为 IP 地址池 pool2 应用配置：

```
$ oc apply -f ipaddresspool2.yaml
```

3. 创建 BGP peer1。

   a. 创建一个文件，如 bgppeer1.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer1
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

   b. 应用 BGP peer 的配置：

```
$ oc apply -f bgppeer1.yaml
```

4. 创建 BGP peer2。

   a. 创建一个文件，如 bgppeer2.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer2
spec:
  peerAddress: 10.0.0.2
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

   b. 应用 BGP peer2 的配置：

```
$ oc apply -f bgppeer2.yaml
```

5. 创建 BGP 广告 1。

   a. 创建一个文件，如 bgpadvertisement1.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool1
  peers:
    - peer1
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

   b. 应用配置：

```
$ oc apply -f bgpadvertisement1.yaml
```

6. 创建 BGP 广告 2。

   a. 创建一个文件，如 bgpadvertisement2.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool2
```

```
peers:
  - peer2
communities:
  - 65535:65282
aggregationLength: 32
aggregationLengthV6: 128
localPref: 100
```

b. 应用配置：

```
$ oc apply -f bgpadvertisement2.yaml
```

34.6.4. 通过网络 VRF 公开服务

您可以通过在网络接口上将 VRF 与 BGP 对等点关联，通过虚拟路由和转发(VRF)实例公开服务。

重要

通过 BGP 对等点上的 VRF 公开服务只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

通过使用网络接口上的 VRF 通过 BGP 对等点公开服务，您可以将流量隔离到服务，配置独立路由决策，并在网络接口上启用多租户支持。

注意

通过一个属于网络 VRF 的接口建立 BGP 会话，MetalLB 可以通过该接口公告服务，并让外部流量通过这个接口访问该服务。但是，network VRF 路由表与 OVN-Kubernetes 使用的默认 VRF 路由表不同。因此，流量无法访问 OVN-Kubernetes 网络基础架构。

要启用定向到服务的流量访问 OVN-Kubernetes 网络基础架构，您必须配置路由规则来为网络流量定义下一跃点。如需更多信息，请参阅*附加资源*部分中的"使用 MetalLB 管理对称路由"中的 NodeNetworkConfigurationPolicy 资源。

这些是通过带有 BGP peer 的网络 VRF 公开服务的高级步骤：

1. 定义 BGP peer 并添加网络 VRF 实例。
2. 为 MetalLB 指定 IP 地址池。
3. 为 MetalLB 配置 BGP 路由广告，以使用指定的 IP 地址池以及与 VRF 实例关联的 BGP 对等点公告路由。
4. 部署服务来测试配置。

先决条件

- 已安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录。

- 您定义了一个 NodeNetworkConfigurationPolicy，将虚拟路由和转发(VRF)实例与网络接口关联。有关完成此先决条件的更多信息，请参阅*附加资源*部分。
- 在集群中安装了 MetalLB。

流程

1. 创建 BGPPeer 自定义资源 (CR)：
   a. 创建一个文件，如 frrviavrf.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf (1)
```

(1) 指定要与 BGP peer 关联的网络 VRF 实例。MetalLB 可以公告服务并根据 VRF 中的路由信息做出路由决策。

注意

您必须在 NodeNetworkConfigurationPolicy CR 中配置此网络 VRF 实例。如需更多信息，请参阅*附加资源*。

   b. 运行以下命令，应用 BGP peer 的配置：

```
$ oc apply -f frrviavrf.yaml
```

2. 创建一个 IPAddressPool CR：
   a. 创建一个文件，如 first-pool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
```

   b. 运行以下命令，为 IP 地址池应用配置：

```
$ oc apply -f first-pool.yaml
```

3. 创建 BGPAdvertisement CR：

a. 创建一个文件，如 first-adv.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
   - first-pool
  peers:
   - frrviavrf (1)
```

(1) 在本例中，MetalLB 将来自 first-pool IP 地址池的 IP 地址范围公告给 frrviavrf BGP peer。

b. 运行以下命令，应用 BGP 公告的配置：

```
$ oc apply -f first-adv.yaml
```

4. 创建一个 Namespace, Deployment, 和 Service CR:

a. 创建一个文件，如 deploy-service.yaml，其内容类似以下示例：

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server
  namespace: test
spec:
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
      - name: server
        image: registry.redhat.io/ubi9/ubi
        ports:
        - name: http
          containerPort: 30100
        command: ["/bin/sh", "-c"]
        args: ["sleep INF"]
---
apiVersion: v1
kind: Service
```

```
  metadata:
    name: server1
    namespace: test
  spec:
    ports:
    - name: http
      port: 30100
      protocol: TCP
      targetPort: 30100
    selector:
      app: server
    type: LoadBalancer
```

b. 运行以下命令，为命名空间、部署和服务应用配置：

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

验证

1. 运行以下命令来识别 MetalLB speaker pod：

```
$ oc get -n metallb-system pods -l component=speaker
```

输出示例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-c6c5f   6/6     Running   0          69m
```

2. 运行以下命令，在 speaker pod 中验证 BGP 会话的状态是否为 Established，替换变量以匹配您的配置：

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf
<vrf_name> neigh"
```

输出示例

```
BGP neighbor is 192.168.30.1, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 192.168.30.1, local router ID 192.168.30.71
  BGP state = Established, up for 04:20:09

...
```

3. 运行以下命令验证该服务是否已正确公告：

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf
<vrf_name> ipv4"
```

其他资源

- 关于虚拟路由和转发
- 示例：带有 VRF 实例网络配置策略的网络接口

- 配置出口服务
- 使用 MetalLB 管理对称路由

34.6.5. BGP 对等配置示例

34.6.5.1. 示例：限制节点连接到 BGP peer

您可以指定节点选择器字段来控制哪些节点可以连接到 BGP 对等点。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-nodesel
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  nodeSelectors:
  - matchExpressions:
    - key: kubernetes.io/hostname
      operator: In
      values: [compute-1.example.com, compute-2.example.com]
```

34.6.5.2. 示例：为 BGP peer 指定 BFD 配置集

您可以指定一个 BFD 配置集，以与 BGP 对等点关联。BFD 复杂的 BGP 通过单独提供与 BGP 间通信故障的更快速检测。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-peer-bfd
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  holdTime: "10s"
  bfdProfile: doc-example-bfd-profile-full
```

注意

删除双向转发检测 (BFD) 配置集并删除添加到边框网关协议 (BGP) 对等资源中的 bfdProfile 不会禁用 BFD。相反，BGP 对等点开始使用默认的 BFD 配置集。要从 BGP peer 资源禁用 BFD，请删除 BGP 对等配置，并在没有 BFD 配置集的情况下重新创建它。如需更多信息，请参阅 BZ#2050824。

34.6.5.3. 示例：为双栈网络指定 BGP 对等点

要支持双栈网络，请为 IPv4 添加一个 BGP peer 自定义资源，并为 IPv6 添加一个 BGP peer 自定义资源。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv4
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64500
  myASN: 64500
---
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500
```

34.6.6. 后续步骤

- 将服务配置为使用 MetalLB

34.7. 配置社区别名

作为集群管理员，您可以配置一个社区别名并在不同的广告中使用它。

34.7.1. 关于社区自定义资源

community 自定义资源是社区的一个别名集合。用户可使用 BGPAdvertis 定义广告 ipAddressPools 时使用的命名别名。下表中描述了 community 自定义资源的字段。

注意

community CRD 仅适用于 BGPAdvertisement。

表 34.6. MetalLB 社区自定义资源

字段	类型	描述
metadata.name	**string**	指定 **community** 的名称。
metadata.namespace	**string**	指定 **community** 的命名空间。指定 MetalLB Operator 使用的同一命名空间。
spec.communities	**string**	指定可在 BGPAdvertisements 中使用的 BGP 社区别名列表。社区别名由名称（别名）和值（数字：number）组成。通过引用 **spec.communities** 字段中的别名名称，将 BGPAdvertisement 链接到社区别名。

表 34.7. CommunityAlias

字段	类型	描述
name	**string**	**community** 的别名名称。
value	**string**	与给定名称对应的 BGP **community** 值。

34.7.2. 使用 BGP 广告和社区别名配置 MetalLB

按如下所示配置 MetalLB，以便 BGP 协议广告 IPAddressPool，并将社区别名设置为 NO_ADVERTISE 社区的数字值。

在以下示例中，对等 BGP 路由器 doc-example-peer-community 接收一个 203.0.113.200/32 路由，以及一个 fc00:f853:ccd:e799::1/128 路由，每个 load-balancer IP 地址都分配给服务。使用 NO_ADVERTISE 社区配置了一个社区别名。

先决条件

- **安装 OpenShift CLI（oc）。**
- **以具有 cluster-admin 特权的用户身份登录。**

流程

1. **创建 IP 地址池。**
   a. **创建一个文件，如 ipaddresspool.yaml，其内容类似以下示例：**

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-community
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
```

   b. **应用 IP 地址池的配置：**

```
$ oc apply -f ipaddresspool.yaml
```

2. **创建名为 community1 的社区别名。**

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
```

```
communities:
  - name: NO_ADVERTISE
    value: '65535:65282'
```

3. 创建一个名为 doc-example-bgp-peer 的 BGP peer。

   a. 创建一个文件，如 bgppeer.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-bgp-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

   b. 应用 BGP peer 的配置：

```
$ oc apply -f bgppeer.yaml
```

4. 创建一个带有社区别名的 BGP 广告。

   a. 创建一个文件，如 bgpadvertisement.yaml，内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgp-community-sample
  namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - NO_ADVERTISE 1
  ipAddressPools:
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

   1 在这里指定 CommunityAlias.name，而不是社区自定义资源 (CR) 名称。

   b. 应用配置：

```
$ oc apply -f bgpadvertisement.yaml
```

34.8. 配置 METALLB BFD 配置集

作为集群管理员，您可以添加、修改和删除双向检测(BFD)配置集。MetalLB Operator 使用 BFD 配置集自定义资源来识别哪个 BGP 会话使用 BFD 来单独提供比 BGP 更快地提供的路径故障检测。

34.8.1. 关于 BFD 配置集自定义资源

下表中描述了 BFD 配置集自定义资源的字段。

表 34.8. BFD 配置集自定义资源

字段	类型	描述
metadata.name	**string**	指定 BFD 配置集自定义资源的名称。
metadata.namespace	**string**	指定 BFD 配置集自定义资源的命名空间。
spec.detectMultiplier	**整数**	指定确定数据包丢失的检测倍数。远程传输间隔乘以这个值来确定连接丢失检测计时器。 例如，当本地系统的检测倍数设置为 **3**，而远程系统会将传输间隔设置为 **300** 时，本地系统仅在没有接收数据包的 **900** ms 后才会检测故障。 范围为 **2** 到 **255**。默认值为 **3**。
spec.echoMode	**布尔值**	指定回显传输模式。如果您不使用分布式 BFD，则回显传输模式仅在 peer 也是 FRR 时才可以正常工作。默认值为 **false**，而回显传输模式被禁用。 启用回显传输模式时，请考虑增加控制数据包的传输间隔，以减少带宽使用量。例如，考虑将传输间隔增加到 **2000** 毫秒。
spec.echoInterval	**整数**	指定此系统用来发送和接收回显数据包的最小传输间隔（较少的）。范围为 **10** 到 **60000**。默认值为 **50** ms。
spec.minimumTtl	**整数**	指定传入控制数据包的最低预期 TTL。此字段只适用于多跃点会话。 设置最小 TTL 的目的是使数据包验证要求更加严格，并避免从其他会话接收控制数据包。 默认值为 **254**，表示系统在该系统和对等点之间仅需要一个跃点。
spec.passiveMode	**布尔值**	指定会话是否标记为主动或者被动。被动会话不会尝试启动连接。相反，被动会话会等待来自 peer 的控制数据包，然后再开始回复。 当您有一个作为星星网络的中央节点，并且您希望发送不需要系统发送的控制数据包时，如果您有一个路由器将会话标记为被动。 默认值为 **false**，并将会话标记为 active。
spec.receiveInterval	**整数**	指定此系统可以接收控制数据包的最低间隔。范围为 **10** 到 **60000**。默认值为 **300** ms。

字段	类型	描述
spec.transmitInterval	整数	指定此系统用来发送控制数据包的最小传输间隔（较少的）。范围为 10 到 60000。默认值为 300 ms。

34.8.2. 配置 BFD 配置集

作为集群管理员，您可以添加 BFD 配置集，并配置 BGP 对等点来使用配置集。BFD 仅提供比 BGP 快于 BGP 的路径故障检测速度。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个文件，如 bfdprofile.yaml，其内容如下：

```
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: doc-example-bfd-profile-full
  namespace: metallb-system
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
  echoMode: false
  passiveMode: true
  minimumTtl: 254
```

2. 为 BFD 配置集应用配置：

```
$ oc apply -f bfdprofile.yaml
```

34.8.3. 后续步骤

- 将 BGP peer 配置为使用 BFD 配置集。

34.9. 将服务配置为使用 METALLB

作为集群管理员，当添加类型为 LoadBalancer 的服务时，您可以控制 MetalLB 如何分配 IP 地址。

34.9.1. 请求特定 IP 地址

与其他一些负载均衡器实施一样，MetalLB 接受服务规格中的 spec.loadBalancerIP 字段。

如果请求的 IP 地址位于任何地址池中，MetalLB 会分配请求的 IP 地址。如果请求的 IP 地址不在任何范围内，MetalLB 会报告警告。

特定 IP 地址的服务 YAML 示例

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <ip_address>
```

如果 MetalLB 无法分配请求的 IP 地址，服务报告的 EXTERNAL-IP 会报告 <pending>，运行 oc describe service <service_name> 会包括一个类似以下示例的事件。

当 MetalLB 无法分配请求的 IP 地址时的示例

```
  ...
Events:
  Type     Reason          Age    From             Message
  ----     ------          ----   ----             -------
  Warning  AllocationFailed  3m16s  metallb-controller  Failed to allocate IP for "default/invalid-request": "4.3.2.1" is not allowed in config
```

34.9.2. 从特定池中请求 IP 地址

要从特定范围分配 IP 地址，但您不关注特定的 IP 地址，您可以使用 metallb.universe.tf/address-pool 注解从指定地址池中请求 IP 地址。

来自特定池的 IP 地址的服务 YAML 示例

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

如果您为 <address_pool_name> 指定的地址池不存在，MetalLB 会尝试从允许自动分配的池中分配 IP 地址。

34.9.3. 接受任何 IP 地址

默认情况下，地址池配置为允许自动分配。MetalLB 从这些地址池中分配 IP 地址。

若要接受任何为自动分配配置的池的 IP 地址，不需要特殊注释或配置。

接受任何 IP 地址的服务 YAML 示例

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

34.9.4. 共享特定 IP 地址

默认情况下，服务不共享 IP 地址。但是，如果您需要在单个 IP 地址上并置服务，可以通过向服务添加 metallb.universe.tf/allow-shared-ip 注解来启用选择性 IP 共享。

```
apiVersion: v1
kind: Service
metadata:
  name: service-http
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc" 1
spec:
  ports:
    - name: http
      port: 80 2
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> 3
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 4
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc"
spec:
  ports:
    - name: https
```

```
    port: 443
    protocol: TCP
    targetPort: 8080
  selector:
    <label_key>: <label_value>
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7
```

1. 为 metallb.universe.tf/allow-shared-ip 注解指定相同的值。此值被称为 *共享键（sharing key）*。
2. 为服务指定不同的端口号。
3. 如果您必须指定 externalTrafficPolicy: local，以便服务将流量发送到同一组 pod，则指定相同的 pod 选择器。如果您使用 cluster 外部流量策略，则 pod 选择器不需要相同。
4. 可选：如果您指定了上述三个项目，MetalLB 可能会将服务在同一 IP 地址上并置。若要确保服务共享 IP 地址，请指定要共享的 IP 地址。

默认情况下，Kubernetes 不允许多协议负载均衡器服务。此限制通常会导致无法运行需要同时侦听 TCP 和 UDP 的服务（如 DNS）。要解决 Kubernetes 使用 MetalLB 的这一限制，请创建两个服务：

- 对于一个服务，请为第二个服务指定 TCP 和 指定 UDP。
- 在两个服务中，指定相同的 pod 选择器。
- 指定相同的共享密钥和 spec.loadBalancerIP 值，以将 TCP 和 UDP 服务在同一 IP 地址上并置。

34.9.5. 使用 MetalLB 配置服务

您可以将负载平衡服务配置为使用地址池中的外部 IP 地址。

先决条件

- 安装 OpenShift CLI (oc)。
- 安装 MetalLB Operator 并启动 MetalLB。
- 至少配置一个地址池。
- 配置网络，将流量从客户端路由到集群的主机网络。

流程

1. 创建一个 <service_name>.yaml 文件。在文件中，确保将 spec.type 字段设置为 LoadBalancer。
   有关如何请求 MetalLB 分配给服务的外部 IP 地址的信息，请参考示例。
2. 创建服务：

```
$ oc apply -f <service_name>.yaml
```

输出示例

```
service/<service_name> created
```

验证

- 描述该服务：

```
$ oc describe service <service_name>
```

输出示例

```
Name:                     <service_name>
Namespace:                default
Labels:                   <none>
Annotations:              metallb.universe.tf/address-pool: doc-example  (1)
Selector:                 app=service_name
Type:                     LoadBalancer  (2)
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.105.237.254
IPs:                      10.105.237.254
LoadBalancer Ingress:     192.168.100.5  (3)
Port:                     <unset>  80/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30550/TCP
Endpoints:                10.244.0.50:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:  (4)
  Type    Reason        Age              From            Message
  ----    ------        ----             ----            -------
  Normal  nodeAssigned  32m (x2 over 32m)  metallb-speaker  announcing from node "
<node_name>"
```

(1) 如果您从特定池请求 IP 地址，则会显示该注释。

(2) 服务类型必须表示 LoadBalancer。

(3) 如果服务被正确分配，load-balancer ingress 字段会指示外部 IP 地址。

(4) events 字段显示分配给声明外部 IP 地址的节点名称。如果出现错误，Event 字段会指示错误的原因。

34.10. 使用 METALLB 管理对称路由

作为集群管理员，您可以通过从 MetalLB、NMState 和 OVN-Kubernetes 实现功能来有效地管理带有多个主机接口的 MetalLB 负载均衡器服务后面的 pod 的流量。通过结合此上下文中的这些功能，您可以提供对称路由、流量隔离，并支持具有不同网络上的客户端及重叠 CIDR 地址。

要实现此功能，了解如何使用 MetalLB 实施虚拟路由和转发(VRF)实例，并配置出口服务。

重要

使用带有 MetalLB 和出口服务的 VRF 实例配置对称流量只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

34.10.1. 使用 MetalLB 管理对称路由的挑战

当您将 MetalLB 与多个主机接口搭配使用时，MetalLB 通过主机上的所有可用接口公开并宣布服务。这可能会造成与网络隔离、非对称返回流量和重叠 CIDR 地址相关的挑战。

确保返回流量达到正确的客户端的一个选项是使用静态路由。但是，使用这个解决方案，MetalLB 无法隔离服务，然后通过不同的接口声明每个服务。另外，静态路由需要手动配置，并在添加远程站点时需要维护。

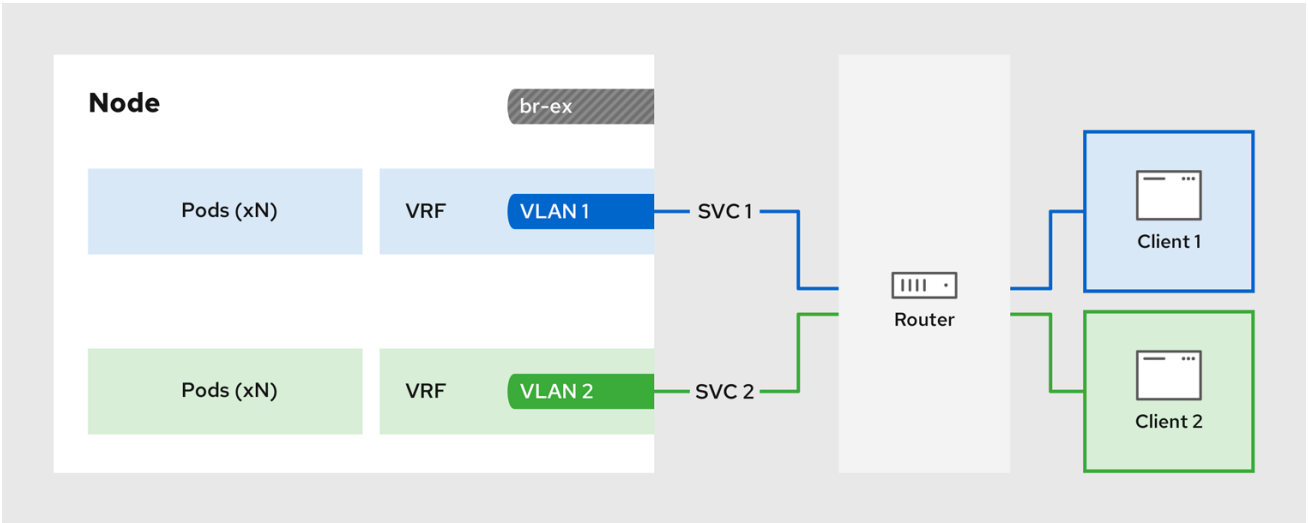
在实现 MetalLB 服务时，对称路由的进一步挑战是外部系统预期应用程序的源和目标 IP 地址相同的情况。OpenShift Container Platform 的默认行为是将主机网络接口的 IP 地址分配为来自 pod 的流量的源 IP 地址。这在多个主机接口中存在问题。

您可以通过实施组合来自 MetalLB、NMState 和 OVN-Kubernetes 的功能的配置来解决这些挑战。

34.10.2. 使用带有 MetalLB 的 VRF 管理对称路由概述

您可以使用 NMState 在主机上配置 VRF 实例、将 VRF 实例与 MetalLB BGPPeer 资源关联，以及为出口流量配置出口服务来克服实施对称路由的挑战。

图 34.2. 使用带有 MetalLB 的 VRF 管理对称路由的网络概述



配置过程涉及三个阶段：

1.定义 VRF 和路由规则

- 配置 NodeNetworkConfigurationPolicy 自定义资源 (CR)，将 VRF 实例与网络接口关联。
- 使用 VRF 路由表直接入口和出口流量。

2.将 VRF 链接到 MetalLB BGPPeer

- 配置 MetalLB BGPPeer 资源，以使用网络接口上的 VRF 实例。
- 通过将 BGPPeer 资源与 VRF 实例关联，指定的网络接口成为 BGP 会话的主接口，MetalLB 通过这个接口公告服务。

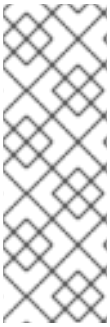
3.配置出口服务

- 配置出口服务，为出口流量选择与 VRF 实例关联的网络。
- 可选：将出口服务配置为使用 MetalLB 负载均衡器服务的 IP 地址作为出口流量的源 IP。

34.10.3. 使用带有 MetalLB 的 VRF 配置对称路由

您可以为需要同一入口和出口网络路径的 MetalLB 服务后面的应用程序配置对称网络路由。

这个示例将 VRF 路由表与 MetalLB 和出口服务相关联，以便为 LoadBalancer 服务后面的 pod 启用对称路由。



注意

- 如果您在 EgressService CR 中使用 sourceIPBy: "LoadBalancerIP" 设置，您必须在 BGPAdvertisement 自定义资源 (CR) 中指定负载均衡器节点。
- 在使用带有 gatewayConfig.routingViaHost 规格设置为 true 的 OVN-Kubernetes 的集群中，您可以使用 sourceIPBy: "Network" 设置。另外，如果您使用 sourceIPBy: "Network" 设置，您必须在使用网络 VRF 实例配置的节点上调度应用程序工作负载。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 创建一个 NodeNetworkConfigurationPolicy CR 来定义 VRF 实例：
   a. 创建一个文件，如 node-network-vrf.yaml，其内容类似以下示例：

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy (1)
spec:
  nodeSelector:
    vrf: "true" (2)
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens4vrf (3)
        type: vrf (4)
        state: up
```

```
    vrf:
      port:
        - ens4 5
      route-table-id: 2 6
  routes: 7
    config:
      - destination: 0.0.0.0/0
        metric: 150
        next-hop-address: 192.168.130.1
        next-hop-interface: ens4
        table-id: 2
  route-rules: 8
    config:
      - ip-to: 172.30.0.0/16
        priority: 998
        route-table: 254
      - ip-to: 10.132.0.0/14
        priority: 998
        route-table: 254
```

1. 策略的名称。
2. 这个示例将策略应用到带有 vrf:true 标签的所有节点。
3. 接口的名称。
4. 接口的类型。这个示例创建了一个 VRF 实例。
5. VRF 附加到的节点接口。
6. VRF 的路由表 ID 的名称。
7. 定义网络路由的配置。next-hop-address 字段定义路由的下一跃点的 IP 地址。next-hop-interface 字段定义路由的传出接口。在本例中，VRF 路由表是 2，它引用您在 EgressService CR 中定义的 ID。
8. 定义额外的路由规则。ip-to 字段必须与 Cluster Network CIDR 和 Service Network CIDR 匹配。您可以运行以下命令来查看这些 CIDR 地址规格的值： oc describe network.config/cluster。

b. 运行以下命令来应用策略：

```
$ oc apply -f node-network-vrf.yaml
```

2. 创建 BGPPeer 自定义资源 (CR)：

a. 创建一个文件，如 frr-via-vrf.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
```

```
peerASN: 200
peerAddress: 192.168.130.1
vrf: ens4vrf 1
```

1 指定要与 BGP peer 关联的 VRF 实例。MetalLB 可以公告服务并根据 VRF 中的路由信息做出路由决策。

b. 运行以下命令，应用 BGP peer 的配置：

```
$ oc apply -f frr-via-vrf.yaml
```

3. 创建一个 IPAddressPool CR：

a. 创建一个文件，如 first-pool.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
```

b. 运行以下命令，为 IP 地址池应用配置：

```
$ oc apply -f first-pool.yaml
```

4. 创建 BGPAdvertisement CR：

a. 创建一个文件，如 first-adv.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
   - first-pool
  peers:
   - frrviavrf 1
  nodeSelectors:
   - matchLabels:
       egress-service.k8s.ovn.org/test-server1: "" 2
```

1 在本例中，MetalLB 将来自 first-pool IP 地址池的 IP 地址范围公告给 frrviavrf BGP peer。

2 在本例中，EgressService CR 将出口流量的源 IP 地址配置为使用负载均衡器服务 IP 地址。因此，您必须指定负载均衡器节点返回流量，以便为来自 pod 的流量使用相同的返回路径。

b. 运行以下命令，应用 BGP 公告的配置：

```
$ oc apply -f first-adv.yaml
```

5. 创建一个 EgressService CR：

a. 创建一个文件，如 egress-service.yaml，其内容类似以下示例：

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: server1 1
  namespace: test 2
spec:
  sourceIPBy: "LoadBalancerIP" 3
  nodeSelector:
    matchLabels:
      vrf: "true" 4
  network: "2" 5
```

1. 指定出口服务的名称。EgressService 资源的名称必须与您要修改的负载均衡器服务的名称匹配。

2. 指定出口服务的命名空间。EgressService 的命名空间必须与您要修改的负载均衡器服务的命名空间匹配。egress 服务是命名空间范围的。

3. 这个示例将 LoadBalancer 服务入口 IP 地址分配为出口流量的源 IP 地址。

4. 如果为 sourceIPBy 规格指定 LoadBalancer，则单一节点处理 LoadBalancer 服务流量。在本例中，只有标签 vrf: "true" 的节点才能处理服务流量。如果没有指定节点，OVN-Kubernetes 会选择一个 worker 节点来处理服务流量。当选择节点时，OVN-Kubernetes 以以下格式标记节点： egress-service.k8s.ovn.org/<svc_namespace>-<svc_name>: ""。

5. 指定出口流量的路由表。

b. 运行以下命令，为出口服务应用配置：

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

验证

1. 运行以下命令，验证您可以访问 MetalLB 服务后运行的 pod 的应用程序端点：

```
$ curl <external_ip_address>:<port_number> 1
```

1. 更新外部 IP 地址和端口号，以适合您的应用程序端点。

2. 可选：如果您将 LoadBalancer 服务入口 IP 地址指定为出口流量的源 IP 地址，请使用 tcpdump 等工具分析外部客户端接收的数据包，以此验证此配置。

其他资源

- 关于虚拟路由和转发
- 通过网络 VRF 公开服务
- 示例：带有 VRF 实例网络配置策略的网络接口
- 配置出口服务

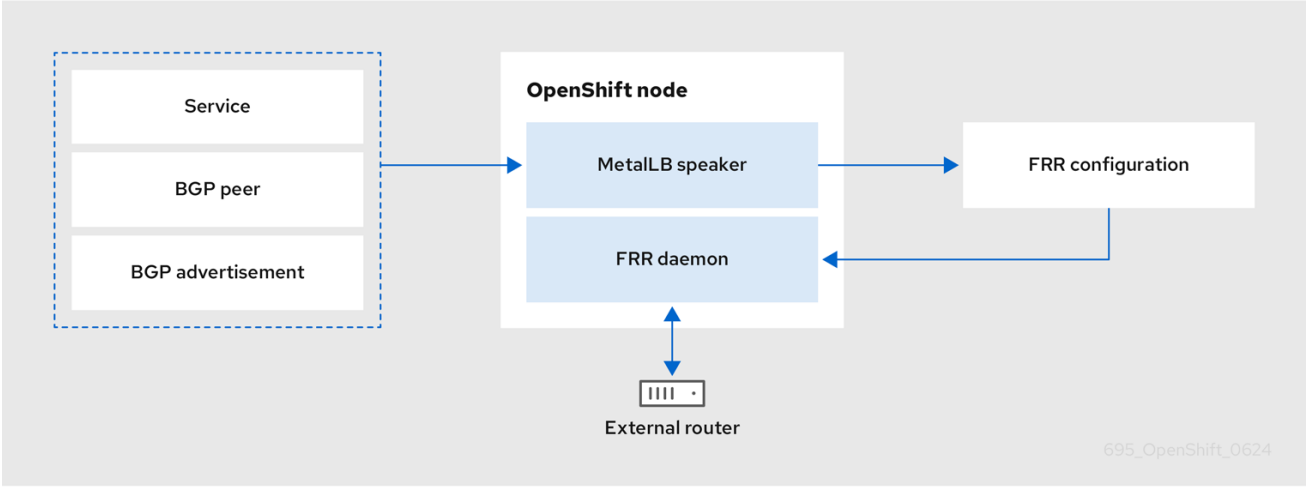
34.11. 配置 METALLB 和 FRR-K8S 的集成

重要

FRRConfiguration 自定义资源只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

FRRouting (FRR) 是一个免费的开源互联网路由协议套件，用于 Linux 和 UNIX 平台。FRR-K8s 是基于 Kubernetes 的 DaemonSet，它以 Kubernetes 兼容的方式公开 FRR API 的子集。作为集群管理员，您可以使用 FRRConfiguration 自定义资源 (CR) 将 MetalLB 配置为使用 FRR-K8s 作为后端。您可以使用它来提供 FRR 服务，例如接收路由。如果您使用 FRR-K8s 作为后端运行 MetalLB，MetalLB 会生成与应用的 MetalLB 配置对应的 FRR-K8s 配置。



34.11.1. 激活 MetalLB 和 FRR-K8s 的集成

以下流程演示了如何激活 FRR-K8s 作为 MetalLB 的后端。

先决条件

- **您已在裸机硬件上安装了集群。**
- **已安装 OpenShift CLI(oc)。**
- **您已以具有 cluster-admin 权限的用户身份登录。**

流程

- 将 MetalLB CR 的bgpBackend 字段设置为 frr-k8s，如下例所示：

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  bgpBackend: frr-k8s
```

34.11.2. FRR 配置

您可以创建多个 FRRConfiguration CR，以便在 MetalLB 中使用 FRR 服务。MetalLB 生成一个 FRRConfiguration 对象，它 FRR-K8s 与所有用户创建的所有配置合并。

例如，您可以配置 FRR-K8s，以接收给定邻居公告的所有前缀。以下示例将 FRR-K8s 配置为接收带有主机 172.18.0.5 的 BGPPeer 公告的所有前缀：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
 name: test
 namespace: metallb-system
spec:
 bgp:
   routers:
   - asn: 64512
     neighbors:
     - address: 172.18.0.5
       asn: 64512
       toReceive:
        allowed:
           mode: all
```

您还可以将 FRR-K8s 配置为始终阻止一组前缀，而不考虑应用的配置。这可用于避免到可能造成集群故障的 pod 或 ClusterIP CIDR 的路由。以下示例阻止了一组前缀 192.168.1.0/24 ：

MetalLB CR 示例

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  bgpBackend: frr-k8s
  frrk8sConfig:
    alwaysBlock:
    - 192.168.1.0/24
```

您可以将 FRR-K8s 设置为阻止 Cluster Network CIDR 和 Service Network CIDR。您可以运行以下命令来查看这些 CIDR 地址规格的值：

```
$ oc describe network.config/cluster
```

34.11.3. 配置 FRRConfiguration CRD

以下部分提供了使用 FRRConfiguration 自定义资源 (CR) 的参考示例。

34.11.3.1. router 字段

您可以使用 router 字段配置多个路由器，每个虚拟路由和转发 (VRF) 资源对应一个。对于每个路由器，您必须定义自主系统号 (ASN)。

您还可以定义要连接的边框网关协议 (BGP) 邻居列表，如下例所示：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
        port: 180
      - address: 172.18.0.6
        asn: 4200000000
        port: 179
```

34.11.3.2. toAdvertise 字段

默认情况下，FRR-K8s 不会公告配置为路由器配置的前缀。要公告它们，您可以使用 toAdvertise 字段。

您可以公告前缀的子集，如下例所示：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
```

```
      port: 180
      toAdvertise:
        allowed:
          prefixes: (1)
          - 192.168.2.0/24
      prefixes:
        - 192.168.2.0/24
        - 192.169.2.0/24
```

(1) 广告一个前缀子集。

以下示例演示了如何公告所有前缀：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.30.0.3
        asn: 4200000000
        ebgpMultiHop: true
        port: 180
        toAdvertise:
          allowed:
            mode: all (1)
      prefixes:
        - 192.168.2.0/24
        - 192.169.2.0/24
```

(1) 公告所有前缀。

34.11.3.3. toReceive 字段

默认情况下，FRR-K8s 不处理邻居公告的任何前缀。您可以使用 toReceive 字段来处理此类地址。

您可以为前缀的子集配置，如下例所示：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
```

```
    routers:
    - asn: 64512
      neighbors:
      - address: 172.18.0.5
          asn: 64512
          port: 179
          toReceive:
            allowed:
              prefixes:
              - prefix: 192.168.1.0/24
              - prefix: 192.169.2.0/24
                ge: 25 (1)
                le: 28 (2)
```

(1) (2) 如果前缀长度小于或等于 le 的前缀长度，且大于或等于 ge 的前缀长度，则应用前缀。

以下示例将 FRR 配置为处理声明的所有前缀：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
      - address: 172.18.0.5
          asn: 64512
          port: 179
          toReceive:
            allowed:
              mode: all
```

34.11.3.4. bgp 字段

您可以使用 bgp 字段定义各种 BFD 配置集，并将它们与邻居关联。在以下示例中，BFD 备份 BGP 会话，FRR 可以检测链接失败：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
      neighbors:
```

```
    - address: 172.30.0.3
      asn: 64512
      port: 180
      bfdProfile: defaultprofile
  bfdProfiles:
    - name: defaultprofile
```

34.11.3.5. nodeSelector 字段

默认情况下，FRR-K8s 将配置应用到守护进程运行的所有节点。您可以使用 nodeSelector 字段指定要应用配置的节点。例如：

FRRConfiguration CR 示例

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
    - asn: 64512
  nodeSelector:
    labelSelector:
    foo: "bar"
```

FRRConfiguration 自定义资源的字段在下表中描述：

表 34.9. MetalLB FRRConfiguration 自定义资源

字段	类型	描述
spec.bgp.routers	**数组**	指定 FRR 来配置的路由器（每个 VRF 一个）。
spec.bgp.routers.asn	**整数**	用于会话本地端的自主系统编号。
spec.bgp.routers.id	**string**	指定 **bgp** 路由器的 ID。
spec.bgp.routers.vrf	**string**	指定用于建立来自此路由器会话的主机 vrf。
spec.bgp.routers.neighbors	**数组**	指定要建立 BGP 会话的邻居。
spec.bgp.routers.neighbors.asn	**整数**	指定用于会话本地端的自主系统编号。

字段	类型	描述
spec.bgp.routers.neighbors.address	**string**	指定要建立会话的 IP 地址。
spec.bgp.routers.neighbors.port	**整数**	指定建立会话时要结束的端口。默认值为 179。
spec.bgp.routers.neighbors.password	**string**	指定用来建立 BGP 会话的密码。**password** 和 **PasswordSecret** 是互斥的。
spec.bgp.routers.neighbors.passwordSecret	**string**	指定邻居的身份验证 secret 的名称。secret 必须是 "kubernetes.io/basic-auth" 类型，并且与 FRR-K8s 守护进程位于同一个命名空间中。密钥 "password" 将密码存储在 secret 中。**password** 和 **PasswordSecret** 是互斥的。
spec.bgp.routers.neighbors.holdTime	**duration**	根据 RFC4271 指定请求的 BGP 保留时间。默认为 180s。
spec.bgp.routers.neighbors.keepaliveTime	**duration**	根据 RFC4271 指定请求的 BGP keepalive 时间。默认值为 **60s**。
spec.bgp.routers.neighbors.connectTime	**duration**	指定 BGP 在连接尝试到邻居之间等待的时间。
spec.bgp.routers.neighbors.ebgpMultiHop	**布尔值**	指明 BGPPeer 是否离开了多跃点。
spec.bgp.routers.neighbors.bfdProfile	**string**	指定用于与 BGP 会话关联的 BFD 会话的 BFD Profile 名称。如果没有设置，则不会设置 BFD 会话。
spec.bgp.routers.neighbors.toAdvertise.allowed	**数组**	表示要公告给邻居的前缀列表，以及相关的属性。
spec.bgp.routers.neighbors.toAdvertise.allowed.prefixes	**字符串数组**	指定要公告到邻居的前缀列表。此列表必须与您在路由器中定义的前缀匹配。

字段	类型	描述
spec.bgp.routers.neighbors.toAdvertise.allowed.mode	**string**	指定处理前缀时要使用的模式。您可以将 **filtered** 设置为只允许前缀列表中的前缀。您可以设置为 **all**，以允许路由器上配置的所有前缀。
spec.bgp.routers.neighbors.toAdvertise.withLocalPref	**数组**	指定与公告的本地首选项关联的前缀。您必须在允许公告的前缀中指定与本地首选项关联的前缀。
spec.bgp.routers.neighbors.toAdvertise.withLocalPref.prefixes	**字符串数组**	指定与本地首选项关联的前缀。
spec.bgp.routers.neighbors.toAdvertise.withLocalPref.localPref	**整数**	指定与前缀关联的本地首选项。
spec.bgp.routers.neighbors.toAdvertise.withCommunity	**数组**	指定与公告的 BGP 社区关联的前缀。您必须在您要公告的前缀列表中包含与本地首选项关联的前缀。
spec.bgp.routers.neighbors.toAdvertise.withCommunity.prefixes	**字符串数组**	指定与社区关联的前缀。
spec.bgp.routers.neighbors.toAdvertise.withCommunity.community	**string**	指定与前缀关联的社区。
spec.bgp.routers.neighbors.toReceive	**数组**	指定要从邻居接收的前缀。
spec.bgp.routers.neighbors.toReceive.allowed	**数组**	指定要从邻居接收的信息。
spec.bgp.routers.neighbors.toReceive.allowed.prefixes	**数组**	指定来自邻居的前缀。

字段	类型	描述
spec.bgp.routers.neighbors.toReceive.allowed.mode	**string**	指定处理前缀时要使用的模式。当设置为 **filtered** 时，只允许 **prefixes** 列表中的前缀。当设置为 **all** 时，允许路由器上配置的所有前缀。
spec.bgp.routers.neighbors.disableMP	**布尔值**	禁用 MP BGP 以防止它将 IPv4 和 IPv6 路由划分为不同的 BGP 会话。
spec.bgp.routers.prefixes	**字符串数组**	指定从此路由器实例公告的所有前缀。
spec.bgp.bfdProfiles	**数组**	指定配置邻居时要使用的 bfd 配置集列表。
spec.bgp.bfdProfiles.name	**string**	要在配置的其他部分中引用的 BFD 配置集的名称。
spec.bgp.bfdProfiles.receiveInterval	**整数**	指定此系统可以接收控制数据包的最小间隔（以毫秒为单位）。默认值为 **300ms**。
spec.bgp.bfdProfiles.transmitInterval	**整数**	指定排除 jitter 的最小传输间隔，此系统希望用来发送 BFD 控制数据包（以毫秒为单位）。默认值为 **300ms**。
spec.bgp.bfdProfiles.detectMultiplier	**整数**	配置检测倍数以确定数据包丢失。要确定连接丢失检测计时器，请将远程传输间隔乘以这个值。
spec.bgp.bfdProfiles.echoInterval	**整数**	配置此系统可以处理的最小 echo receive transfer-interval （以毫秒为单位）。默认值为 **50ms**。
spec.bgp.bfdProfiles.echoMode	**布尔值**	启用或禁用回显传输模式。这个模式默认为禁用，在多跃点设置中不支持。
spec.bgp.bfdProfiles.passiveMode	**布尔值**	将会话标记为被动。被动会话不会尝试启动连接，并在开始回复前等待来自对等的控制数据包。

字段	类型	描述
spec.bgp.bfdProfiles.MinimumTtl	**整数**	仅限多跃点会话。为传入的 BFD 控制数据包配置最低预期 TTL。
spec.nodeSelector	**string**	限制尝试应用此配置的节点。如果指定，则只有标签与指定选择器匹配的节点才会应用配置。如果没有指定，则所有节点都会尝试应用此配置。
status	**string**	定义 FRRConfiguration 的观察状态。

34.11.4. FRR-K8s 如何合并多个配置

如果多个用户添加选择同一节点的配置，FRR-K8s 会合并配置。每个配置都只能扩展其他配置。这意味着，可以添加新的邻居到路由器，或向邻居公告额外的前缀，但不能删除由其他配置添加的组件。

34.11.4.1. 配置冲突

某些配置可能会导致冲突，从而导致错误，例如：

- **同一路由器的不同 ASN（在同一 VRF 中）**
- **同一邻居的不同 ASN（具有相同 IP / 端口）**
- **多个带有相同名称的 BFD 配置集，但不同的值**

当守护进程为节点找到无效的配置时，它会将配置报告为无效，并恢复到之前有效的 FRR 配置。

34.11.4.2. 合并

在合并时，可以执行以下操作：

- **将您要公告的 IP 集合扩展到邻居。**
- **使用 IP 集合添加额外的邻居。**
- **扩展您要关联社区的 IP 集合。**
- **允许邻居的传入路由。**

每个配置都必须自包含。例如，通过利用来自其他配置的前缀，不允许在路由器部分中定义的前缀。

如果要应用的配置兼容，则合并可以正常工作，如下所示：

- **frr-K8s 结合了所有路由器。**
- **frr-K8s 合并每个路由器的所有前缀和邻居。**
- **frr-K8s 合并每个邻居的所有过滤器。**

注意

不太严格的过滤会优先于更严格的过滤。例如，接受某些前缀的过滤会优先于不接受任何前缀的过滤，接受所有前缀的过滤优先于接受部分前缀的过滤。

34.12. METALLB 日志记录、故障排除和支持

如果您需要对 MetalLB 配置进行故障排除，请查看以下部分来了解常用命令。

34.12.1. 设置 MetalLB 日志记录级别

MetalLB 在带有默认设置 info 的容器中使用 FRRouting(FRR)会生成大量日志。您可以通过设置 logLevel 来控制生成的日志的详细程度，如下例所示。

通过将 logLevel 设置为 debug 来深入了解 MetalLB，如下所示：

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI(oc)。

流程

1. 创建一个文件，如 setdebugloglevel.yaml，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

2. 应用配置：

```
$ oc replace -f setdebugloglevel.yaml
```

注意

使用 oc replace 可以被理解为，metallb CR 已被创建，您在此处更改了日志级别。

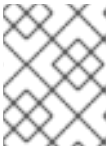
3. 显示 speaker pod 的名称：

```
$ oc get -n metallb-system pods -l component=speaker
```

输出示例

```
NAME                READY   STATUS    RESTARTS   AGE
speaker-2m9pm       4/4     Running   0          9m19s
```

```
speaker-7m4qw        3/4     Running   0        19s
speaker-szlmx        4/4     Running   0        9m19s
```



注意

重新创建发言人和控制器 Pod，以确保应用更新的日志记录级别。对于 MetalLB 的所有组件，会修改日志记录级别。

4. 查看 **speaker** 日志：

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

输出示例

{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 / amd64","level":"info","msg":"MetalLB speaker starting (commit 3d052535, branch main)","ts":"2022-05-17T09:55:05Z","version":""}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"ens4","level":"info","msg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"ens4","level":"info","msg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"tun0","level":"info","msg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"tun0","level":"info","msg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
I0517 09:55:06.515686   95 request.go:665] Waited for 1.026500832s due to client-side throttling, not priority and fairness, request: GET:https://172.30.0.1:443/apis/operators.coreos.com/v1alpha1?timeout=32s
{"Starting Manager":"(MISSING)","caller":"k8s.go:389","level":"info","ts":"2022-05-17T09:55:08Z"}
{"caller":"speakerlist.go:310","level":"info","msg":"node event - forcing sync","node addr":"10.0.128.4","node event":"NodeJoin","node name":"ci-ln-qb8t3mb-72292-7s7rh-worker-a-vvznj","ts":"2022-05-17T09:55:08Z"}
{"caller":"service_controller.go:113","controller":"ServiceReconciler","enqueueing":"openshift-kube-controller-manager-operator/metrics","epslice":"{\"metadata\":{\"name\":\"metrics-xtsxr\",\"generateName\":\"metrics-\",\"namespace\":\"openshift-kube-controller-manager-operator\",\"uid\":\"ac6766d7-8504-492c-9d1e-4ae8897990ad\",\"resourceVersion\":\"9041\",\"generation\":4,\"creationTimestamp\":\"2022-05-17T07:16:53Z\",\"labels\":{\"app\":\"kube-controller-manager-operator\",\"endpointslice.kubernetes.io/managed-by\":\"endpointslice-controller.k8s.io\",\"kubernetes.io/service-name\":\"metrics\"},\"annotations\":{\"endpoints.kubernetes.io/last-change-trigger-time\":\"2022-05-17T07:21:34Z\"},\"ownerReferences\":[{\"apiVersion\":\"v1\",\"kind\":\"Service\",\"name\":\"metrics\",\"uid\":\"0518eed3-6152-42be-b566-0bd00a60faf8\",\"controller\":true,\"blockOwnerDeletion\":true}],\"managedFields\":[{\"manager\":\"kube-controller-manager\",\"operation\":\"Update\",\"apiVersion\":\"discovery.k8s.io/v1\",\"time\":\"2022-05-17T07:20:02Z\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:addressType\":{},\"f:endpoints\":{},\"f:metadata\":{\"f:annotations\":{\".\":{},\"f:endpoints.kubernetes.io/last-change-trigger-time\":{}},\"f:generateName\":{},\"f:labels\":{\".\":{},\"f:app\":{},\"f:endpointslice.kubernetes.io/managed-by\":{},\"f:kubernetes.io/service-name\":{}},\"f:ownerReferences\":{\".\":{},\"k:{\\\"uid\\\":\\\"0518eed3-6152-42be-b566-0bd00a60faf8\\\"}\":{}}},\"f:ports\":{}}}]},\"addressType\":\"IPv4\",\"endpoints\":[{\"addresses\":[\"10.129.0.7\"],\"conditions\":{\"ready\":true,\"serving\":true,\"terminating\":false},\"targetRef\":

{\"kind\":\"Pod\",\"namespace\":\"openshift-kube-controller-manager-operator\",\"name\":\"kube-controller-manager-operator-6b98b89ddd-8d4nf\",\"uid\":\"dd5139b8-e41c-4946-a31b-1a629314e844\",\"resourceVersion\":\"9038\"},\"nodeName\":\"ci-ln-qb8t3mb-72292-7s7rh-master-0\",\"zone\":\"us-central1-a\"}],\"ports\":[{\"name\":\"https\",\"protocol\":\"TCP\",\"port\":8443}]}","level":"debug","ts":"2022-05-17T09:55:08Z"}

5. 查看 **FRR** 日志：

```
$ oc logs -n metallb-system speaker-7m4qw -c frr
```

输出示例

```
Started watchfrr
2022/05/17 09:55:05 ZEBRA: client 16 says hello and bids fair to announce only bgp routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 31 says hello and bids fair to announce only vnc routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 38 says hello and bids fair to announce only static routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 43 says hello and bids fair to announce only bfd routes vrf=0
2022/05/17 09:57:25.089 BGP: Creating Default VRF, AS 64500
2022/05/17 09:57:25.090 BGP: dup addr detect enable max_moves 5 time 180 freeze disable freeze_time 0
2022/05/17 09:57:25.090 BGP: bgp_get: Registering BGP instance (null) to zebra
2022/05/17 09:57:25.090 BGP: Registering VRF 0
2022/05/17 09:57:25.091 BGP: Rx Router Id update VRF 0 Id 10.131.0.1/32
2022/05/17 09:57:25.091 BGP: RID change : vrf VRF default(0), RTR ID 10.131.0.1
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF br0
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ens4
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr 10.0.128.4/32
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr fe80::c9d:84da:4d86:5618/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF lo
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ovs-system
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF tun0
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr 10.131.0.1/23
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr fe80::40f1:d1ff:feb6:5322/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2da49fed
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2da49fed addr fe80::24bd:d1ff:fec1:d88/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2fa08c8c
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2fa08c8c addr fe80::6870:ff:fe96:efc8/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth41e356b7
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth41e356b7 addr fe80::48ff:37ff:fede:eb4b/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth1295c6e2
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth1295c6e2 addr fe80::b827:a2ff:feed:637/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth9733c6dc
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth9733c6dc addr
```

```
fe80::3cf4:15ff:fe11:e541/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth336680ea
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth336680ea addr
fe80::94b1:8bff:fe7e:488c/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vetha0a907b7
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vetha0a907b7 addr
fe80::3855:a6ff:fe73:46c3/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf35a4398
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf35a4398 addr
fe80::40ef:2fff:fe57:4c4d/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf831b7f4
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf831b7f4 addr
fe80::f0d9:89ff:fe7c:1d32/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vxlan_sys_4789
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vxlan_sys_4789 addr
fe80::80c1:82ff:fe4b:f078/64
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Timer (start timer expire).
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] BGP_Start (Idle->Connect), fd -1
2022/05/17 09:57:26.094 BGP: Allocated bnc 10.0.0.1/32(0)(VRF default) peer
0x7f807f7631a0
2022/05/17 09:57:26.094 BGP: sendmsg_zebra_rnh: sending cmd
ZEBRA_NEXTHOP_REGISTER for 10.0.0.1/32 (vrf VRF default)
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Waiting for NHT
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Connect
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Idle to Connect
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] TCP_connection_open_failed (Connect-
>Active), fd -1
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Active
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Connect to Active
2022/05/17 09:57:26.094 ZEBRA: rnh_register msg from client bgp: hdr->length=8,
type=nexthop vrf=0
2022/05/17 09:57:26.094 ZEBRA: 0: Add RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: Evaluate RNH, type Nexthop (force)
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: NH has become unresolved
2022/05/17 09:57:26.094 ZEBRA: 0: Client bgp registers for RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 BGP: VRF default(0): Rcvd NH update 10.0.0.1/32(0) - metric 0/0
#nhops 0/0 flags 0x6
2022/05/17 09:57:26.094 BGP: NH update for 10.0.0.1/32(0)(VRF default) - flags 0x6
chgflags 0x0 - evaluate paths
2022/05/17 09:57:26.094 BGP: evaluate_paths: Updating peer (10.0.0.1(VRF default)) status
with NHT
2022/05/17 09:57:30.081 ZEBRA: Event driven route-map update triggered
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-out
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-in
2022/05/17 09:57:31.104 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.104 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring
2022/05/17 09:57:31.105 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.105 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring
```

34.12.1.1. FRRouting(FRR)日志级别

下表描述了 FRR 日志记录级别。

表 34.10. 日志级别

日志级别	描述
all	为所有日志记录级别提供所有日志信息。
debug	这些信息有助于相关人员进行问题诊断。设置为 **debug**，以提供详细的故障排除信息。
info	提供始终应记录的信息，但在正常情况下，不需要用户干预。这是默认的日志记录级别。
warn	任何可能导致 **MetalLB** 行为不一致的情况。通常 **MicalLB** 会自动从这类错误中恢复。
错误	对 **MetalLB** 功能有验证影响的重大错误。这些错误通常需要管理员干预才能修复。
none	关闭所有日志。

34.12.2. BGP 故障排除问题

红帽支持在 speaker Pod 的容器中使用 FRRouting(FRR)的 BGP 实施。作为集群管理员，如果您需要对 BGP 配置问题进行故障排除，您需要在 FRR 容器中运行命令。

先决条件

- **您可以使用具有 cluster-admin 角色的用户访问集群。**
- **已安装 OpenShift CLI(oc)。**

流程

1. **显示 speaker pod 的名称：**

```
$ oc get -n metallb-system pods -l component=speaker
```

输出示例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-66bth   4/4     Running   0          56m
speaker-gvfnf   4/4     Running   0          56m
...
```

2. **显示 FRR 的运行配置：**

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show running-config"
```

输出示例

```
Building configuration...

Current configuration:
!
frr version 7.5.1_git
frr defaults traditional
hostname some-hostname
log file /etc/frr/frr.log informational
log timestamp precision 3
service integrated-vtysh-config
!
router bgp 64500 (1)
 bgp router-id 10.0.1.2
 no bgp ebgp-requires-policy
 no bgp default ipv4-unicast
 no bgp network import-check
 neighbor 10.0.2.3 remote-as 64500 (2)
 neighbor 10.0.2.3 bfd profile doc-example-bfd-profile-full (3)
 neighbor 10.0.2.3 timers 5 15
 neighbor 10.0.2.4 remote-as 64500
 neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full
 neighbor 10.0.2.4 timers 5 15
 !
 address-family ipv4 unicast
  network 203.0.113.200/30 (4)
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
 !
 address-family ipv6 unicast
  network fc00:f853:ccd:e799::/124
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
!
route-map 10.0.2.3-in deny 20
!
route-map 10.0.2.4-in deny 20
!
ip nht resolve-via-default
!
ipv6 nht resolve-via-default
!
line vty
!
bfd
 profile doc-example-bfd-profile-full
  transmit-interval 35
  receive-interval 35
```

```
 passive-mode
 echo-mode
 echo-interval 35
 minimum-ttl 10
!
!
end
```

1. **router bgp 部分指示 MetalLB 的 ASN。**
2. **确认添加的每个 BGP peer 自定义资源都有一个 neighbor <ip-address> remote-as <peer-ASN> 行。**
3. **如果您配置了 BFD，请确认 BFD 配置集已与正确的 BGP peer 关联，并且 BFD 配置集出现在命令输出中。**
4. **确认 network <ip-address-range> 行与您在地址池自定义资源中指定的 IP 地址范围匹配。**

3. **显示 BGP 概述：**

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp summary"
```

输出示例

```
IPv4 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02            0        1 (1)
10.0.2.4        4      64500         0         0        0    0    0    never       Active        0 (2)

Total number of neighbors 2

IPv6 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02 NoNeg
10.0.2.4        4      64500         0         0        0    0    0    never       Active        0

Total number of neighbors 2
```

1. **确认输出包含您添加的每个 BGP peer 自定义资源的行。**
2. **显示收到的 0 个消息，并且发送的消息指示没有 BGP 会话的 BGP 对等点。检查 BGP 对等点的网络连接和 BGP 配置。**

4. 显示接收地址池的 BGP 对等点：

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp ipv4 unicast 203.0.113.200/30"
```

将 ipv4 替换为 ipv6，以显示接收 IPv6 地址池的 BGP 对等点。将203.0.113.200/30 替换为地址池的 IPv4 或 IPv6 IP 地址范围。

输出示例

```
BGP routing table entry for 203.0.113.200/30
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
  10.0.2.3 1
  Local
    0.0.0.0 from 0.0.0.0 (10.0.1.2)
      Origin IGP, metric 0, weight 32768, valid, sourced, local, best (First path received)
      Last update: Mon Jan 10 19:49:07 2022
```

1 确认输出中包含 BGP peer 的 IP 地址。

34.12.3. BFD 问题故障排除

红帽支持双向转发检测(BFD)实施，在 speaker Pod 中使用 FRRouting(FRR)。BFD 实施依赖于 BFD 对等点，也被配置为带有已建立的 BGP 会话的 BGP 对等点。作为集群管理员，如果您需要排除 BFD 配置问题，则需要在 FRR 容器中运行命令。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI(oc)。

流程

1. 显示 speaker pod 的名称：

```
$ oc get -n metallb-system pods -l component=speaker
```

输出示例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-66bth   4/4     Running   0          26m
speaker-gvfnf   4/4     Running   0          26m
...
```

2. 显示 BFD 对等点：

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
```

输出示例

```
Session count: 2
SessionId  LocalAddress           PeerAddress           Status
=========  ============          ===========           ======
3909139637 10.0.1.2              10.0.2.3              up  <.>
```

确认 PeerAddress 列包含每个 BFD 对等点。如果输出没有列出输出要包含的 BFD peer IP 地址，并与 peer 对 BGP 连接性进行故障排除。如果状态字段显示 down，请检查在节点和对等点间的链接和设备连接。您可以使用 oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}' 命令确定 speaker pod 的节点名称。

34.12.4. BGP 和 BFD 的 MetalLB 指标

OpenShift Container Platform 为 MetalLB 捕获以下与 BGP 对等点和 BFD 配置集相关的指标。

表 34.11. MetalLB BFD 指标

Name	描述
metallb_bfd_control_packet_input	统计从每个 BFD peer 接收的 BFD 控制数据包的数量。
metallb_bfd_control_packet_output	统计发送到每个 BFD 对等点的 BFD 控制数据包的数量。
metallb_bfd_echo_packet_input	统计从每个 BFD 对等点接收的 BFD 回显数据包的数量。
metallb_bfd_echo_packet_output	计算发送到每个 BFD 的 BFD 回显数据包的数量。
metallb_bfd_session_down_events	统计 BFD 会话进入 **down** 状态的次数。
metallb_bfd_session_up	指示与 BFD 对等点的连接状态。**1** 表示会话状态为 **up**，**0** 表示会话为 **down**。
metallb_bfd_session_up_events	统计 BFD 会话进入 **up** 状态的次数。
metallb_bfd_zebra_notifications	统计每个 BFD Zebra 通知的数量。

表 34.12. MetalLB BGP 指标

Name	描述
metallb_bgp_announced_prefixes_total	计算公告给 BGP 对等的负载均衡器 IP 地址前缀的数量。术语 *前缀(prefix)* 和 *聚合路由(aggregated route)* 具有相同的含义。

Name	描述
metallb_bgp_session_up	指示与 BGP 对等点的连接状态。**1** 表示会话状态为 **up**，**0** 表示会话为 **down**。
metallb_bgp_updates_total	计算发送到每个 BGP 对等点的 BGP 更新消息数量。
metallb_bgp_opens_sent	计算发送到每个 BGP 对等点的 BGP 打开消息数量。
metallb_bgp_opens_received	计算从每个 BGP 对等点接收的 BGP 打开消息的数量。
metallb_bgp_notifications_sent	计算发送到每个 BGP 对等点的 BGP 通知消息数量。
metallb_bgp_updates_total_received	计算从每个 BGP 对等点接收的 BGP 更新消息数量。
metallb_bgp_keepalives_sent	计算发送到每个 BGP 对等点的 BGP keepalive 消息数量。
metallb_bgp_keepalives_received	计算从每个 BGP 对等点接收的 BGP keepalive 消息数量。
metallb_bgp_route_refresh_sent	计算发送到每个 BGP 对等点的 BGP 路由刷新消息的数量。
metallb_bgp_total_sent	计算发送到每个 BGP 对等点的 BGP 消息总数。
metallb_bgp_total_received	计算从每个 BGP 对等点接收的 BGP 消息总数。

其他资源

- **有关使用监控仪表板的信息，请参阅 查询指标。**

34.12.5. 关于收集 MetalLB 数据

您可以使用 oc adm must-gather CLI 命令来收集有关集群、MetalLB 配置和 MetalLB Operator 的信息。以下功能和对象与 MetalLB 和 MetalLB Operator 关联：

- **在其中部署 MetalLB Operator 的命名空间和子对象**
- **所有 MetalLB Operator 自定义资源定义(CRD)**

oc adm must-gather CLI 命令会收集红帽用来实施 BGP 和 BFD 的 FRRouting(FRR)的以下信息：

- **/etc/frr/frr.conf**
- **/etc/frr/frr.log**

- /etc/frr/daemons 配置文件
- /etc/frr/vtysh.conf

上述列表中的日志和配置文件从每个 speaker pod 中的 frr 容器收集。

除了日志和配置文件外，oc adm must-gather CLI 命令还会从以下 vtysh 命令收集输出：

- show running-config
- show bgp ipv4
- show bgp ipv6
- show bgp neighbor
- show bfd peer

运行 oc adm must-gather CLI 命令时不需要额外的配置。

其他资源

- 收集集群数据

第 35 章 将二级接口指标与网络附加关联

35.1. 为监控扩展二级网络指标

二级设备或接口用于不同目的。为了对采用相同分类的二级设备的指标数据进行汇总，需要有一个方法来对它们进行分类。

公开的指标会包括接口，但不会指定接口的起始位置。当没有其他接口时，这可以正常工作。但是，如果添加了二级接口，则很难使用指标，因为只使用接口名称识别接口比较困难。

在添加二级接口时，它们的名称取决于添加它们的顺序，不同的二级接口可能属于不同的网络，并可用于不同的目的。

通过使用 pod_network_name_info，可以使用标识接口类型的额外信息来扩展当前的指标。这样，就可以聚合指标，并为特定接口类型添加特定的警告。

网络类型使用相关的 NetworkAttachmentDefinition 名称生成，该名称也用于区分不同类别的次网络。例如，属于不同网络或使用不同的 CNI 的不同接口使用不同的网络附加定义名称。

35.1.1. 网络指标守护进程

网络指标守护进程是收集并发布与网络相关的指标的守护进程组件。

kubelet 已经发布了您可以观察到的网络相关指标。这些指标是：

- container_network_receive_bytes_total
- container_network_receive_errors_total
- container_network_receive_packets_total
- container_network_receive_packets_dropped_total
- container_network_transmit_bytes_total
- container_network_transmit_errors_total
- container_network_transmit_packets_total
- container_network_transmit_packets_dropped_total

这些指标中的标签包括：

- Pod 名称
- Pod 命名空间
- 接口名称（比如 eth0）

这些指标在为 pod 添加新接口之前（例如通过 Multus ）可以正常工作。在添加了新接口后，无法清楚地知道接口名称代表什么。

interface 标签指向接口名称，但它不知道接口的作用是什么。在有多个不同接口的情况下，无法了解您监控的指标代表什么网络。

现在引入了新的 pod_network_name_info 可以帮助解决这个问题。

35.1.2. 带有网络名称的指标

此 daemonset 发布一个 pod_network_name_info 指标，固定值为 0:

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="nadnamespace/firstNAD",pod="podname"} 0
```

使用 Multus 所添加的注解生成网络名称标签。它是网络附加定义所属命名空间的连接，加上网络附加定义的名称。

新的指标本身不会提供很多值，但与网络相关的 container_network_* 指标一起使用，可以为二集网络的监控提供更好的支持。

通过使用类似以下的 promql 查询，可以获取包含值的新指标，以及从 k8s.v1.cni.cncf.io/network-status 注解中检索的网络名称：

```
(container_network_receive_bytes_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_errors_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_bytes_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_errors_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name)
```