



OpenShift Container Platform 4.10

网络

配置和管理集群网络

法律通告

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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 了解网络	7
1.1. OPENSIFT CONTAINER PLATFORM DNS	7
1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	7
1.3. OPENSIFT CONTAINER PLATFORM 网络的常见术语表	8
第 2 章 访问主机	11
2.1. 访问安装程序置备的基础架构集群中 AMAZON WEB SERVICES 上的主机	11
第 3 章 网络 OPERATOR 概述	12
3.1. CLUSTER NETWORK OPERATOR	12
3.2. DNS OPERATOR	12
3.3. INGRESS OPERATOR	12
3.4. 外部 DNS OPERATOR	12
3.5. NETWORK OBSERVABILITY OPERATOR	12
第 4 章 OPENSIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR	13
4.1. CLUSTER NETWORK OPERATOR	13
4.2. 查看集群网络配置	13
4.3. 查看 CLUSTER NETWORK OPERATOR 状态	14
4.4. 查看 CLUSTER NETWORK OPERATOR 日志	14
4.5. CLUSTER NETWORK OPERATOR 配置	14
4.6. 其他资源	20
第 5 章 OPENSIFT CONTAINER PLATFORM 中的 DNS OPERATOR	21
5.1. DNS OPERATOR	21
5.2. 更改 DNS OPERATOR MANAGEMENTSTATE	21
5.3. 控制 DNS POD 放置	22
5.4. 查看默认 DNS	23
5.5. 使用 DNS 转发	23
5.6. DNS OPERATOR 状态	26
5.7. DNS OPERATOR 日志	26
5.8. 设置 COREDNS 日志级别	26
5.9. 设置 COREDNS OPERATOR 的日志级别	27
第 6 章 OPENSIFT CONTAINER PLATFORM 中的 INGRESS OPERATOR	28
6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	28
6.2. INGRESS 配置资产	28
6.3. INGRESS CONTROLLER 配置参数	28
6.4. 查看默认的 INGRESS CONTROLLER	41
6.5. 查看 INGRESS OPERATOR 状态	41
6.6. 查看 INGRESS CONTROLLER 日志	42
6.7. 查看 INGRESS CONTROLLER 状态	42
6.8. 配置 INGRESS CONTROLLER	42
6.9. 其他资源	66
第 7 章 配置 INGRESS CONTROLLER 端点发布策略	67
7.1. INGRESS CONTROLLER 端点发布策略	67
7.2. 其他资源	69
第 8 章 验证到端点的连接	70
8.1. 执行连接健康检查	70
8.2. 连接健康检查实现	70
8.3. PODNETWORKCONNECTIVITYCHECK 对象字段	70

8.4. 验证端点的网络连接	73
第 9 章 更改集群网络的 MTU	78
9.1. 关于集群 MTU	78
9.2. 更改集群 MTU	79
9.3. 其他资源	85
第 10 章 配置节点端口服务范围	86
10.1. 先决条件	86
10.2. 扩展节点端口范围	86
10.3. 其他资源	87
第 11 章 配置 IP 故障转移	88
11.1. IP 故障转移环境变量	89
11.2. 配置 IP 故障转移	90
11.3. 关于虚拟 IP 地址	93
11.4. 配置检查和通知脚本	93
11.5. 配置 VRRP 抢占	95
11.6. 关于 VRRP ID 偏移	96
11.7. 为超过 254 地址配置 IP 故障转移	96
11.8. INGRESSIP 的高可用性	97
11.9. 删除 IP 故障切换	97
第 12 章 在裸机集群中使用流控制传输协议 (SCTP)	100
12.1. 支持 OPENSIFT CONTAINER PLATFORM 上的流控制传输协议 (SCTP)	100
12.2. 启用流控制传输协议 (SCTP)	101
12.3. 验证流控制传输协议 (SCTP) 已启用	102
第 13 章 使用 PTP 硬件	105
13.1. 关于 PTP 硬件	105
13.2. 关于 PTP	105
13.3. 使用 CLI 安装 PTP OPERATOR	106
13.4. 使用 WEB 控制台安装 PTP OPERATOR	107
13.5. 配置 PTP 设备	108
13.6. 常见 PTP OPERATOR 故障排除	124
13.7. PTP 硬件快速事件通知框架	127
第 14 章 外部 DNS OPERATOR	136
14.1. OPENSIFT CONTAINER PLATFORM 中的外部 DNS OPERATOR	136
14.2. 在云供应商上安装外部 DNS OPERATOR	137
14.3. 外部 DNS OPERATOR 配置参数	137
14.4. 在 AWS 上创建 DNS 记录	140
14.5. 在 AZURE 上创建 DNS 记录	141
14.6. 在 GCP 上创建 DNS 记录	143
14.7. 在外部 DNS OPERATOR 上配置集群范围代理	145
第 15 章 网络策略	147
15.1. 关于网络策略	147
15.2. 记录网络策略事件	150
15.3. 创建网络策略	157
15.4. 查看网络策略	160
15.5. 编辑网络策略	161
15.6. 删除网络策略	164
15.7. 为项目定义默认网络策略	164
15.8. 使用网络策略配置多租户隔离	167

第 16 章 多网络	171
16.1. 了解多网络	171
16.2. 配置额外网络	171
16.3. 关于虚拟路由和转发	185
16.4. 配置多网络策略	185
16.5. 将 POD 附加到额外网络	191
16.6. 从额外网络中删除 POD	196
16.7. 编辑额外网络	196
16.8. 删除额外网络	197
16.9. 为 VRF 分配从属网络	198
第 17 章 硬件网络	201
17.1. 关于单根 I/O 虚拟化 (SR-IOV) 硬件网络	201
17.2. 安装 SR-IOV NETWORK OPERATOR	207
17.3. 配置 SR-IOV NETWORK OPERATOR	209
17.4. 配置 SR-IOV 网络设备	214
17.5. 配置 SR-IOV 以太网网络附加	223
17.6. 配置 SR-IOV INFINIBAND 网络附加	229
17.7. 将 POD 添加到额外网络	236
17.8. 配置高性能多播	241
17.9. 使用 DPDK 和 RDMA	243
17.10. 使用 POD 级别绑定	251
17.11. 配置硬件卸载 (OFFLOADING)	254
17.12. 卸载 SR-IOV NETWORK OPERATOR	259
第 18 章 OPENSIFT SDN 默认 CNI 网络供应商	261
18.1. 关于 OPENSIFT SDN 默认 CNI 网络供应商	261
18.2. 为项目配置出口 IP	262
18.3. 为项目配置出口防火墙	268
18.4. 为项目编辑出口防火墙	272
18.5. 为项目编辑出口防火墙	273
18.6. 从项目中删除出口防火墙	274
18.7. 使用出口路由器 POD 的注意事项	274
18.8. 以重定向模式部署出口路由器 POD	277
18.9. 以 HTTP 代理模式部署出口路由器 POD	279
18.10. 以 DNS 代理模式部署出口路由器 POD	282
18.11. 从配置映射配置出口路由器 POD 目的地列表	285
18.12. 为项目启用多播	286
18.13. 为项目禁用多播	289
18.14. 使用 OPENSIFT SDN 配置网络隔离	289
18.15. 配置 KUBE-PROXY	291
第 19 章 OVN-KUBERNETES 默认 CNI 网络供应商	294
19.1. 关于 OVN-KUBERNETES 默认 CONTAINER NETWORK INTERFACE (CNI) 网络供应商	294
19.2. 从 OPENSIFT SDN 集群网络供应商迁移	296
19.3. 回滚到 OPENSIFT SDN 网络供应商	304
19.4. 转换为 IPV4/IPV6 双栈网络	308
19.5. IPSEC 加密配置	310
19.6. 为项目配置出口防火墙	312
19.7. 查看项目的出口防火墙	317
19.8. 为项目编辑出口防火墙	317
19.9. 从项目中删除出口防火墙	318
19.10. 配置出口 IP 地址	318
19.11. 分配出口 IP 地址	325

19.12. 使用出口路由器 POD 的注意事项	326
19.13. 以重定向模式部署出口路由器 POD	328
19.14. 为项目启用多播	333
19.15. 为项目禁用多播	335
19.16. 跟踪网络流	336
19.17. 配置混合联网	339
第 20 章 配置路由	342
20.1. 路由配置	342
20.2. 安全路由	362
第 21 章 配置集群入口流量	367
21.1. 集群入口流量配置概述	367
21.2. 为服务配置 EXTERNALIP	367
21.3. 使用 INGRESS CONTROLLER 配置集群入口流量	373
21.4. 使用负载均衡器配置集群入口流量	378
21.5. 使用网络负载均衡器在 AWS 上配置集群入口流量	382
21.6. 为服务外部 IP 配置 INGRESS 集群流量	385
21.7. 使用 NODEPORT 配置集群入口流量	386
第 22 章 KUBERNETES NMSTATE	390
22.1. 关于 KUBERNETES NMSTATE OPERATOR	390
22.2. 观察节点网络状态	392
22.3. 更新节点网络配置	394
22.4. 对节点网络配置进行故障排除	405
第 23 章 配置集群范围代理	410
23.1. 先决条件	410
23.2. 启用集群范围代理	410
23.3. 删除集群范围代理服务器	412
第 24 章 配置自定义 PKI	413
24.1. 在安装过程中配置集群范围的代理	413
24.2. 启用集群范围代理	414
24.3. 使用 OPERATOR 进行证书注入	416
第 25 章 RHOSP 负载均衡	419
25.1. 使用带有 KURYR SDN 的 OCTAVIA OVN 负载均衡器供应商驱动	419
25.2. 使用 OCTAVIA 为应用程序流量扩展集群	420
25.3. 使用 RHOSP OCTAVIA 为入站流量扩展	422
25.4. 配置外部负载均衡器	424
第 26 章 使用 METALLB 进行负载均衡	427
26.1. 关于 METALLB 和 METALLB OPERATOR	427
26.2. 安装 METALLB OPERATOR	434
26.3. 配置 METALLB 地址池	439
26.4. 配置 METALLB BGP PEER	443
26.5. 配置 METALLB BFD 配置集	447
26.6. 将服务配置为使用 METALLB	449
26.7. METALLB 日志记录、故障排除和支持	453
第 27 章 将二级接口指标与网络附加关联	463
27.1. 为监控扩展二级网络指标	463
第 28 章 NETWORK OBSERVABILITY (网络可观察性)	465
28.1. NETWORK OBSERVABILITY OPERATOR 发行注记	465

28.2. 关于网络可观察性	468
28.3. 安装 NETWORK OBSERVABILITY OPERATOR	469
28.4. OPENSIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR	476
28.5. 配置 NETWORK OBSERVABILITY OPERATOR	478
28.6. NETWORK POLICY	485
28.7. 观察网络流量	486
28.8. 监控 NETWORK OBSERVABILITY OPERATOR	490
28.9. FLOWCOLLECTOR 配置参数	491
28.10. 网络流格式参考	518
28.11. NETWORK OBSERVABILITY 故障排除	523

第 1 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、负载均衡、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。

1.1. OPENSIFT 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。

1.2. OPENSIFT 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** 类型和内部负载均衡）提供了发布 Ingress Controller 端点的方法。

1.2.1. 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。

1.3. OPENSIFT 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)默认网络供应商插件部署。

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。

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 2 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。

2.1. 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 3 章 网络 OPERATOR 概述

OpenShift Container Platform 支持多种类型的网络 Operator。您可以使用这些网络 Operator 管理集群网络。

3.1. 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)默认网络供应商插件部署。如需更多信息，请参阅 [OpenShift Container Platform 中的 Cluster Network Operator](#)。

3.2. 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。如需更多信息，请参阅 [OpenShift Container Platform 中的 DNS Operator](#)。

3.3. INGRESS OPERATOR

创建 OpenShift Container Platform 集群时，集群中运行的 pod 和服务将为每个分配的 IP 地址。IP 地址可以被其他 pod 和服务访问，但外部客户端无法访问。Ingress Operator 实现 Ingress Controller API，并负责启用对 OpenShift Container Platform 集群服务的外部访问。如需更多信息，请参阅 [OpenShift Container Platform 中的 Ingress Operator](#)。

3.4. 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。如需更多信息，请参阅 [了解外部 DNS Operator](#)。

3.5. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator 是一个可选 Operator，它允许集群管理员观察 OpenShift Container Platform 集群的网络流量。Network Observability Operator 使用 eBPF 技术创建网络流。然后，OpenShift Container Platform 信息会增强网络流，并存储在 Loki 中。您可以在 OpenShift Container Platform 控制台中查看和分析所存储的 netflow 信息，以进一步洞察和故障排除。如需更多信息，请参阅 [关于 Network Observability Operator](#)。

第 4 章 OPENSIFT 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) 默认网络供应商插件。

4.1. 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 OpenShift SDN 默认 Container Network Interface (CNI) 网络供应商插件，或部署您在集群安装过程中选择的默认网络供应商插件。

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**。

4.2. 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: ❶
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
    Network Type: OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: ❷
  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>

```

- ❶ **Spec** 字段显示集群网络的已配置状态。
- ❷ **Status** 字段显示集群网络配置的当前状态。

4.3. 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
```

4.4. 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
```

4.5. 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，且这些字段无法更改：

clusterNetwork

从中分配 Pod IP 地址的 IP 地址池。

serviceNetwork

服务的 IP 地址池。

defaultNetwork.type

集群网络供应商，如 OpenShift SDN 或 OVN-Kubernetes。



注意

在集群安装后，您无法修改上一节中列出的字段。

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供应商配置。

4.5.1. Cluster Network Operator 配置对象

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

表 4.1. 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	<p>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：</p> <pre>spec: clusterNetwork: - cidr: 10.128.0.0/19 hostPrefix: 23 - cidr: 10.128.32.0/19 hostPrefix: 23</pre> <p>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。</p>
spec.serviceNetwork	array	<p>服务的 IP 地址块。OpenShift SDN 和 OVN-Kubernetes Container Network Interface(CNI)网络供应商只支持服务网络的一个 IP 地址块。例如：</p> <pre>spec: serviceNetwork: - 172.30.0.0/14</pre> <p>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。</p>

字段	类型	描述
spec.defaultNetwork	object	为集群网络配置 Container Network Interface(CNI)集群网络供应商。
spec.kubeProxyConfig	object	此对象的字段指定 kube-proxy 配置。如果您使用 OVN-Kubernetes 集群网络供应商，则 kube-proxy 配置无效。

defaultNetwork 对象配置

下表列出了 **defaultNetwork** 对象的值：

表 4.2. defaultNetwork 对象

字段	类型	描述
type	字符串	<p>OpenShiftSDN 或 OVNKubernetes。 集群网络供应商是在安装过程中选择的。此值在集群安装后无法更改。</p> <div style="display: flex; align-items: center;">  <div> <p>注意</p> <p>OpenShift Container Platform 默认使用 OpenShift SDN Container Network Interface(CNI)集群网络供应商。</p> </div> </div>
openshiftSDNConfig	object	此对象仅对 OpenShift SDN 集群网络供应商有效。
ovnKubernetesConfig	object	此对象仅对 OVN-Kubernetes 集群网络供应商有效。

OpenShift SDN CNI 集群网络供应商的配置

下表描述了 OpenShift SDN Container Network Interface(CNI)集群网络供应商的配置字段。

表 4.3. openshiftSDNConfig object

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



注意

您只能在集群安装过程中更改集群网络供应商的配置。

OpenShift SDN 配置示例

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

OVN-Kubernetes CNI 集群网络供应商的配置

下表描述了 OVN-Kubernetes CNI 集群网络供应商的配置字段。

表 4.4. ovnKubernetesConfig object

字段	类型	描述
mtu	integer	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	如果存在该字段，则会为集群启用 IPsec。
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

While migrating egress traffic, you can expect some disruption to workloads and service traffic until the Cluster Network Operator (CNO) successfully rolls out the changes.

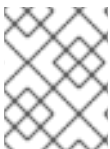
表 4.5. policyAuditConfig object

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。

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

表 4.6. gatewayConfig object

字段	类型	描述
routingViaHost	布尔值	<p>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。</p> <p>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。</p>

**注意**

您只能在集群安装过程中更改集群网络供应商的配置，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: {}
```

kubeProxyConfig object configuration
kubeProxyConfig 对象的值在下表中定义：

表 4.7. kubeProxyConfig object

字段	类型	描述
<code>iptablesSyncPeriod</code>	字符串	<p><code>iptables</code> 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。</p> <div style="display: flex; align-items: flex-start;">  <div> <p>注意</p> <p>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 <code>iptablesSyncPeriod</code> 参数。</p> </div> </div>
<code>proxyArguments.iptables-min-sync-period</code>	array	<p>刷新 <code>iptables</code> 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：</p> <pre>kubeProxyConfig: proxyArguments: iptables-min-sync-period: - 0s</pre>

4.5.2. 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: 1
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: 2
  - 172.30.0.0/16
  defaultNetwork: 3
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period:
        - 0s
```

1 **2** **3** 仅在集群安装过程中配置。

4.6. 其他资源

- [operator.openshift.io](#) API 组中的 **Network** API

第 5 章 OPENSIFT CONTAINER PLATFORM 中的 DNS OPERATOR

DNS Operator 部署并管理 CoreDNS，以为 pod 提供名称解析服务。它在 OpenShift Container Platform 中启用了基于 DNS 的 Kubernetes 服务发现。

5.1. DNS OPERATOR

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
dns      4.1.0-0.11 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 1 个 pod 被设置为 **Available** 状态时，**AVAILABLE** 为 **True**。

5.2. 更改 DNS OPERATOR MANAGEMENTSTATE

DNS 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

- 修改 **managementState** DNS Operator :

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

5.3. 控制 DNS POD 放置

DNS Operator 有两个守护进程集：一个用于 CoreDNS，另一个用于管理 **/etc/hosts** 文件。**/etc/hosts** 的守护进程集必须在每个节点主机上运行，以便为集群镜像 registry 添加条目来支持拉取镜像。安全策略可以禁止节点对之间的通信，这会阻止 CoreDNS 的守护进程集在每个节点上运行。

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。
- 使用具有 **cluster-admin** 权限的用户登陆到集群。

流程

- 要防止某些节点间的通信，请配置 **spec.nodePlacement.nodeSelector** API 字段：

1. 修改名为 **default** 的 DNS Operator 对象：

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

2. 指定在 **spec.nodePlacement.nodeSelector** API 字段中只包含 control plane 节点的节点选择器：

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

- 要允许 CoreDNS 的守护进程集在节点上运行，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**。

5.4. 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，使用 **oc get** 命令：

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

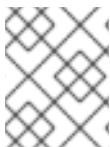
输出示例

```
[172.30.0.0/16]
```

5.5. 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。如果转发区是 OpenShift Container Platform 管理的 Ingress 域，那么上游名称服务器必须为域授权。
- 提供上游 DNS 服务器列表。
- 更改默认转发策略。



注意

默认域的 DNS 转发配置可以同时也在 `/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
```

这允许 Operator 使用基于 **Server** 的额外服务器配置块来创建和更新名为 **dns-default** 的 ConfigMap。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。

DNS 示例

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: foo-server 1
      zones: 2
      - example.com
      forwardPlugin:
        policy: Random 3
        upstreams: 4
        - 1.1.1.1
        - 2.2.2.2:5353
    - name: bar-server
      zones:
      - bar.com
      - example.com
      forwardPlugin:
        policy: Random
        upstreams:
        - 3.3.3.3
        - 4.4.4.4:5454
  upstreamResolvers: 5
    policy: Random 6
    upstreams: 7
    - type: SystemResolvConf 8
    - type: Network
      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** 中子域的定义。集群域 **cluster.local** 不是 **zones** 中的一个有效的 **subdomain**。
- 3** 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**。

- 4 每个 **forwardPlugin** 最多允许 15 个 **upstreams**。
- 5 可选的。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。
- 6 决定选择上游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 可选的。您可以使用它提供上游解析器。
- 8 您可以指定上游的两种类型 - **SystemResolvConf** 和 **Network**。**SystemResolvConf** 将上游配置为使用 `/etc/resolv.conf` 和 **Network** 定义 **Networkresolver**。您可以指定其中一个或两者都指定。
- 9 如果指定类型是 **Network**，则必须提供 IP 地址。**address** 必须是有效的 IPv4 或 IPv6 地址。
- 10 如果指定类型是 **Network**，您可以选择性地提供端口。**port** 必须在 1 到 65535 之间。



注意

如果 **servers** 未定义或无效，则 ConfigMap 只包括默认服务器。

2. 查看 ConfigMap :

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

基于以上 DNS 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](#)。

5.6. 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
```

5.7. 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
```

5.8. 设置 COREDNS 日志级别

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

错误插件会始终被启用。以下 **logLevel** 设置会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
```

5.9. 设置 COREDNS OPERATOR 的日志级别

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**。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
```

第 6 章 OPENSIFT CONTAINER PLATFORM 中的 INGRESS OPERATOR

6.1. OPENSIFT 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** 类型和内部负载均衡）提供了发布 Ingress Controller 端点的方法。

6.2. 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.openshift demos.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。

6.3. 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	<p>domain 是 Ingress Controller 服务的一个 DNS 名称，用于配置多个功能：</p> <ul style="list-style-type: none"> ● 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。 <p>domain 值在所有 Ingress 控制器中需要是唯一的，且不能更新。</p> <p>如果为空，默认值为 ingress.config.openshift.io/cluster.spec.domain。</p>
replicas	<p>replicas 是 Ingress 控制器副本数量。如果没有设置，则默认值为 2。</p>
endpointPublishingStrategy	<p>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。</p> <p>如果没有设置，则默认值基于 infrastructure.config.openshift.io/cluster.status.platform：</p> <ul style="list-style-type: none"> ● AWS: LoadBalancerService （具有外部范围） ● Azure: LoadBalancerService （具有外部范围） ● GCP: LoadBalancerService （具有外部范围） ● 裸机: NodePortService ● 其它: HostNetwork <div data-bbox="619 1330 727 1648" style="display: inline-block; vertical-align: middle;">  </div> <div data-bbox="804 1330 871 1368" style="display: inline-block; vertical-align: middle;"> <p>注意</p> </div> <div data-bbox="804 1400 1430 1554" style="display: inline-block; vertical-align: middle;"> <p>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 16.1 和 16.2，只有使用 Amphora Octavia 供应商时，才能使用这个策略。</p> </div> <div data-bbox="804 1585 1407 1644" style="display: inline-block; vertical-align: middle;"> <p>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。</p> </div> <p>对于大多数平台，可以更新 endpointPublishingStrategy 值。在 GCP 上，您可以配置以下 endpointPublishingStrategy 字段：</p> <ul style="list-style-type: none"> ● loadBalancer.scope ● loadbalancer.providerParameters.gcp.clientAccess ● hostNetwork.protocol ● nodePort.protocol

参数	描述
defaultCertificate	<p>defaultCertificate 的值是一个到包括由 Ingress controller 提供的默认证书的 secret 的指代。当 Routes 没有指定其自身证书时，使用 defaultCertificate。</p> <p>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</p> <p>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。</p> <p>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。</p>
namespaceSelector	<p>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。</p>
routeSelector	<p>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。</p>
nodePlacement	<p>NodePlacement 启用对 Ingress Controller 调度的显式控制。</p> <p>如果没有设置，则使用默认值。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>注意</p> <p>nodePlacement 参数包括两个部分：nodeSelector 和 tolerations。例如：</p> <pre>nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists</pre> </div> </div>

参数	描述
<p>tlsSecurityProfile</p>	<p>tlsSecurityProfile 指定 Ingress Controller 的 TLS 连接的设置。</p> <p>如果没有设置，则默认值基于 <code>apiservers.config.openshift.io/cluster</code> 资源。</p> <p>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。</p> <p>Ingress Controller 的最低 TLS 版本是 1.1，最高 TLS 版本为 1.3。</p> <div data-bbox="518 645 625 775" style="display: inline-block; vertical-align: middle;">  </div> <p style="margin-left: 20px;">注意</p> <p style="margin-left: 20px;">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。</p> <div data-bbox="518 824 625 954" style="display: inline-block; vertical-align: middle;">  </div> <p style="margin-left: 20px;">重要</p> <p style="margin-left: 20px;">Ingress Operator 将 Old 或 Custom 配置集的 TLS1.0 转换为 1.1。</p>
<p>clientTLS</p>	<p>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。</p> <p>clientTLS 具有所需的子字段 spec.clientTLS.clientCertificatePolicy 和 spec.clientTLS.ClientCA。</p> <p>ClientCertificatePolicy 子字段接受以下两个值之一：Required 或 Optional。ClientCA 子字段指定 <code>openshift-config</code> 命名空间中的配置映射。配置映射应包含 CA 证书捆绑包。</p> <p>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。</p>

参数	描述
routeAdmission	<p>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。</p> <p>namespaceOwnership 描述了如何处理跨命名空间的主机名声明。默认为 Strict。</p> <ul style="list-style-type: none">● 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。 <p>wildcardPolicy 描述了 Ingress Controller 如何处理采用通配符策略的路由。</p> <ul style="list-style-type: none">● 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	<p>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。</p> <ul style="list-style-type: none"> ● 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。 <ul style="list-style-type: none"> ○ destination 描述日志消息的目的地。 <ul style="list-style-type: none"> ■ type 是日志的目的地类型： <ul style="list-style-type: none"> ● 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： <ul style="list-style-type: none"> ● 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
<p>httpHeaders</p>	<p>httpHeaders 为 HTTP 标头定义策略。</p> <p>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。</p> <p>默认情况下，策略设置为 Append。</p> <ul style="list-style-type: none"> ● 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。 <p>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。</p> <p>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。</p> <p>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。</p>
<p>httpCompression</p>	<p>httpCompression 定义 HTTP 流量压缩的策略。</p> <ul style="list-style-type: none"> ● 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
<p>httpErrorCodePages</p>	<p>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。</p>

参数	描述
<p>httpCaptureCookies</p>	<p>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。</p> <p>对于您要捕获的任何 Cookie，以下参数必须位于 IngressController 配置中：</p> <ul style="list-style-type: none"> ● 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。 <p>例如：</p> <pre> httpCaptureCookies: - matchType: Exact maxLength: 128 name: MYCOOKIE </pre>
<p>httpCaptureHeaders</p>	<p>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。</p> <p>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：</p> <pre> httpCaptureHeaders: request: - maxLength: 256 name: Connection - maxLength: 128 name: User-Agent response: - maxLength: 256 name: Content-Type - maxLength: 256 name: Content-Length </pre>

参数	描述
tuningOptions	<p>tuningOptions 指定用于调整 Ingress Controller pod 性能的选项。</p> <ul style="list-style-type: none"> ● 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。 ● 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。 ● 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。 ● clientTimeout 指定连接在等待客户端响应时保持打开的时长。如果未设置，则默认超时为 30s。 ● 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。如果未设置，则默认超时为 1s。 ● serverTimeout 指定连接在等待服务器响应时保持打开的时长。如果未设置，则默认超时为 30s。 ● 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。如果未设置，则默认超时为 1s。 ● 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。如果未设置，则默认检查延迟为 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。如果未设置，则默认超时为 1h。

参数	描述
logEmptyRequests	<p>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。</p> <p>LoggingPolicy 类型接受以下两个值之一：</p> <ul style="list-style-type: none"> ● 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	<p>HTTPEmptyRequestsPolicy 描述了在收到请求前发生超时，如何处理 HTTP 连接。此字段允许的值是 Respond 和 Ignore。默认值为 Respond。</p> <p>HTTPEmptyRequestsPolicy 类型接受以下两个值之一：</p> <ul style="list-style-type: none"> ● 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，Ingress Controller 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。 <p>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。</p>



注意

所有参数都是可选的。

6.3.1. 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。

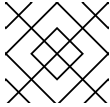

6.3.1.1. 了解 TLS 安全配置集

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 6.1. TLS 安全配置集

profile	描述
---------	----

profile	描述
Old	<p>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。</p> <p>Old 配置集要求最低 TLS 版本 1.0。</p> <div style="display: flex; align-items: center;">  <p>注意</p> </div> <p>对于 Ingress Controller, 最小 TLS 版本从 1.0 转换为 1.1。</p>
Intermediate	<p>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。</p> <p>Intermediate 配置集需要最小 TLS 版本 1.2。</p>
Modern	<p>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。</p> <p>Modern 配置集需要最低 TLS 版本 1.3。</p>
Custom	<p>此配置集允许您定义要使用的 TLS 版本和密码。</p> <div style="background-color: #fff9c4; padding: 10px; margin-top: 10px;"> <div style="display: flex; align-items: center;">  <p>警告</p> </div> <p>使用 Custom 配置集时要谨慎, 因为无效的配置可能会导致问题。</p> </div>



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。

6.3.1.2. 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。

- 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
...

```

6.3.1.3. 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$"
```

6.4. 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
```

6.5. 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
```

6.6. 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>
```

6.7. 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>
```

6.8. 配置 INGRESS CONTROLLER

6.8.1. 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。

6.8.2. 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。

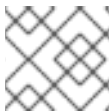
输出示例

—


```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

6.8.3. 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 如果需要不同数量的副本，请更改 **replicas** 值。

6.8.4. 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
```

6.8.5. 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。

6.8.6. Ingress Controller 分片

Ingress Controller 或路由器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。

Ingress Controller 可以使用路由标签或命名空间标签作为分片方法。

6.8.6.1. 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。

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: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net> ❶
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

- ❶ 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
```

6.8.6.2. 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。

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。



警告

如果您部署 Keepalived Ingress VIP，请不要为 **endpointPublishingStrategy** 参数部署带有值 **HostNetwork** 的非默认 Ingress Controller。这样做可能会导致问题。对于 **endpointPublishingStrategy**，使用 **NodePort** 而不是 **HostNetwork**。

流程

1. 编辑 **router-internal.yaml** 文件：

```
# cat router-internal.yaml
```

输出示例

```
apiVersion: v1
items:
- 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
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

- 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
```

6.8.7. 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

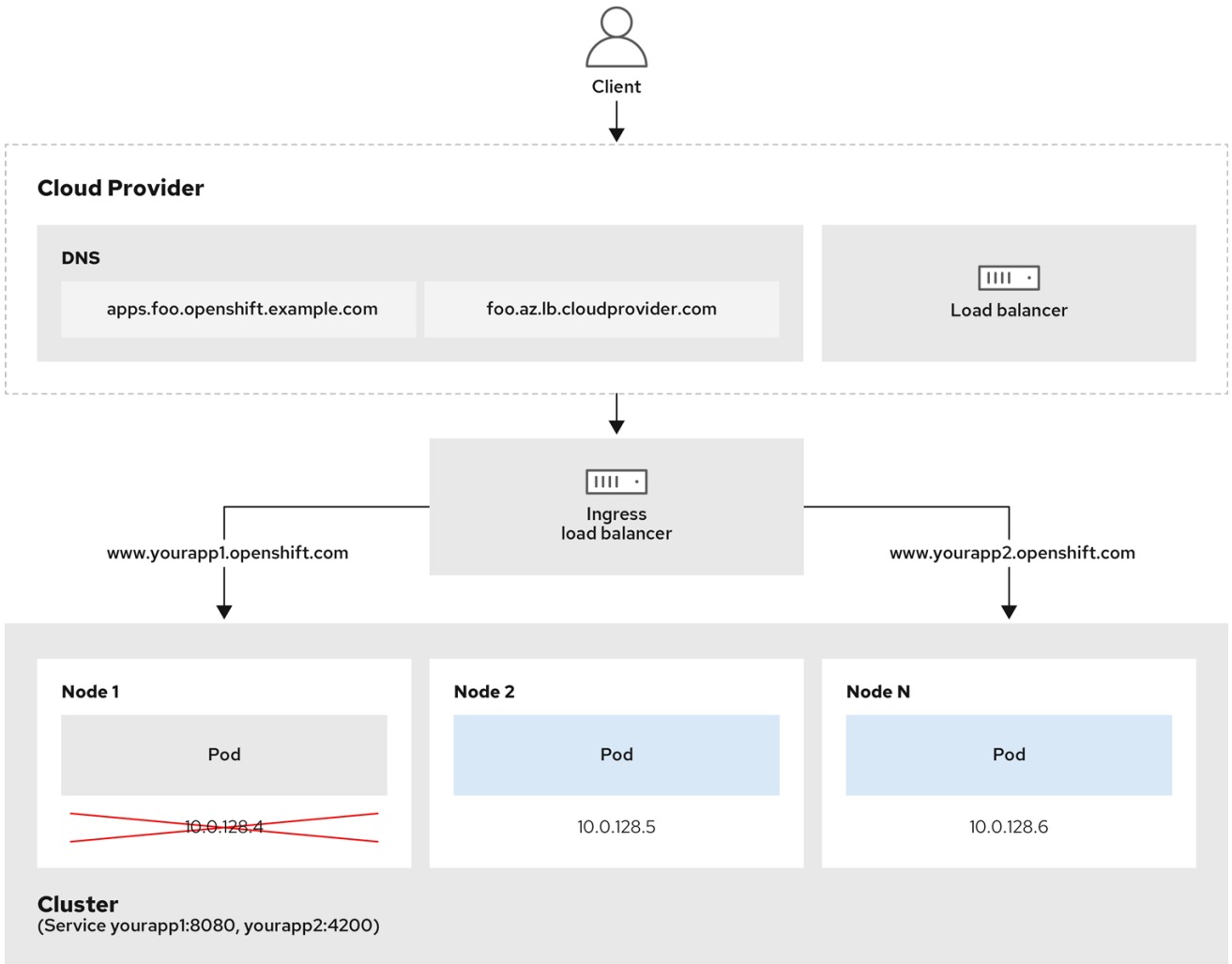
如果云供应商是 Microsoft Azure，则必须至少有一个指向节点的公共负载均衡器。如果不这样做，所有节点都将丢失到互联网的出站连接。



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 6.1. LoadBalancer 图表



202_OpenShift_0222

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> ❶
spec:
  domain: <domain> ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: Internal ❸

```

- ❶ 将 **<name>** 替换为 **IngressController** 对象的名称。
- ❷ 指定控制器发布的应用程序的 **domain**。
- ❸ 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 ❶
```

- ❶ 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
```

6.8.8. 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。

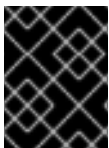
6.8.9. 将集群的默认 Ingress Controller 配置为内部

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
```

6.8.10. 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

```

6.8.11. 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

```

6.8.12. 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。

6.8.13. 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 insecure 或 edge-terminated 路由。



警告

使用带有重新加密路由的 WebSockets，并在 Ingress Controller 上启用 HTTP/2 需要 WebSocket 支持 HTTP/2。通过 HTTP/2 的 websocket 是 HAProxy 2.4 的 WebSocket 功能，目前在 OpenShift Container Platform 中不支持它。



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"
```

6.8.14. 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
```

6.8.15. 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
  appsDomain: <test.example.com>
```

- 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
```

6.8.16. 转换 HTTP 标头的大小写

默认情况下，HAProxy 2.2 使用小写的 HTTP 标头名称，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 4.10 包含 HAProxy 2.2，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

```

- 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

```

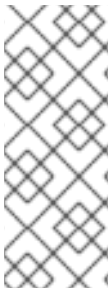
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true 1
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application

```

- 将 `haproxy.router.openshift.io/h1-adjust-case` 设置为 `true`。

6.8.17. 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

- 为 Ingress Controller 配置 `httpCompression` 字段。

- 使用以下命令来编辑 `IngressController` 资源：

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```

- 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"
    ...

```

6.8.18. 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-76bfff66c-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 6.1. 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",service=""} 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
```

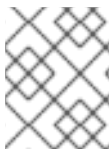
6.8.19. 自定义 HAProxy 错误代码响应页面

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":
{"httpErrorPages":{"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
```

6.9. 其他资源

- [配置自定义 PKI](#)

第 7 章 配置 INGRESS CONTROLLER 端点发布策略

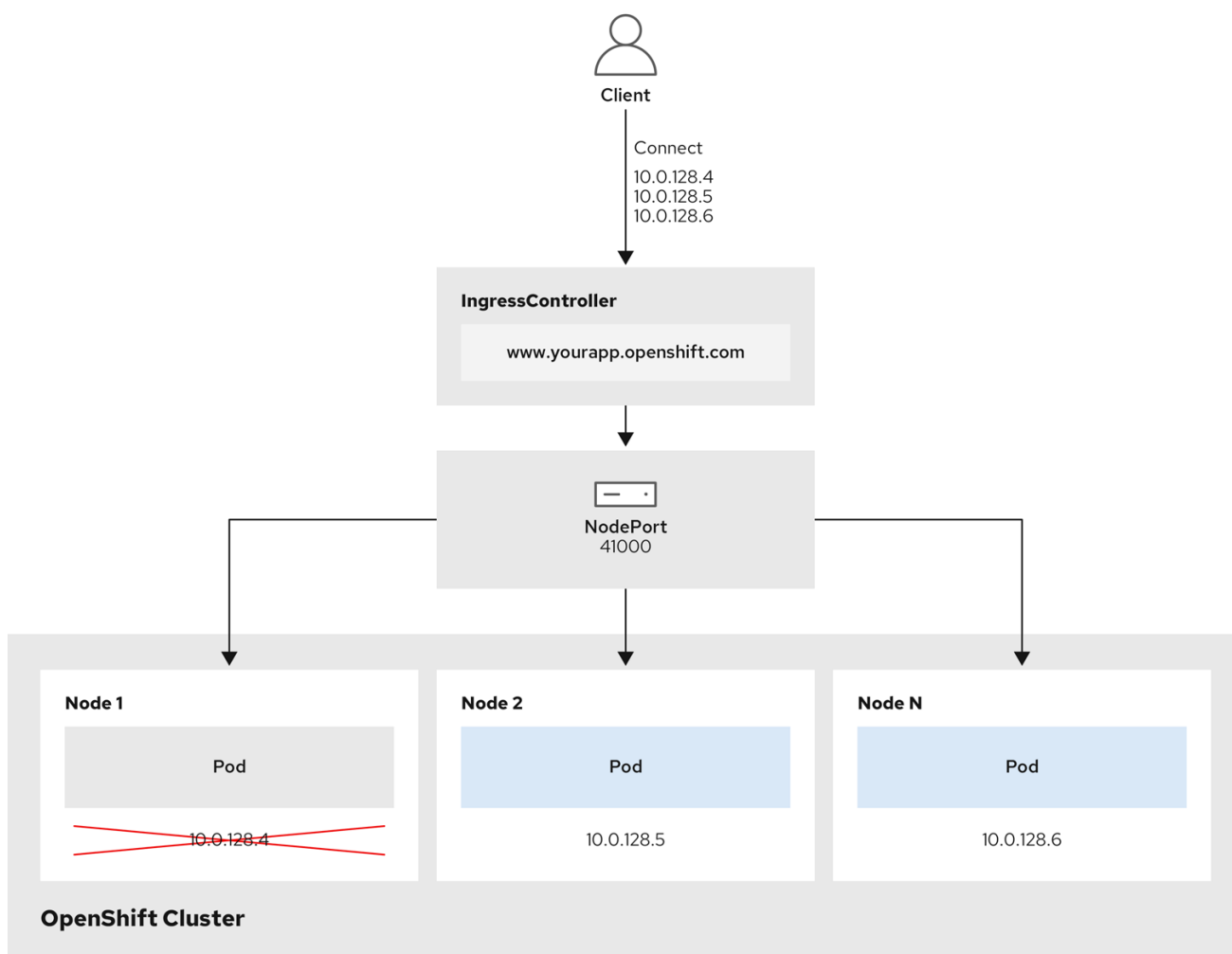
7.1. INGRESS CONTROLLER 端点发布策略

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 7.1. NodePortService 图表



202_OpenShift_0222

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，不需要负载均衡。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。

7.1.1. 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**。

7.1.2. 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**。

7.2. 其他资源

- 如需更多信息，请参阅 [Ingress Controller 配置参数](#)。

第 8 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。

8.1. 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

8.2. 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。

8.3. PODNETWORKCONNECTIVITYCHECK 对象字段

PodNetworkConnectivityCheck 对象字段在下表中描述。

表 8.1. PodNetworkConnectivityCheck 对象字段

字段	类型	描述
----	----	----

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： <ul style="list-style-type: none"> ● 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。
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 8.2. 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 8.3. 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 8.4. 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。

字段	类型	描述
reason	字符串	以可读格式提供状态的原因。这个值是 TCPConnect 、 TCPConnectError 、 DNSResolve 、 DNSErrror 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。

8.4. 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性，如 API 服务器、负载均衡器、服务或 Pod。

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-f76d1-4rgrp-worker-b-6xdmh    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rgrp-worker-c-n8mbf    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rgrp-worker-d-4hnrz    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-
service-cluster                          75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-0 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-1 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-2 74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-openshift-apiserver-
service-cluster                          75m

```

2. 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-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-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-f76d1-4rgrp-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-f76d1-4rgrp-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 9 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 集群网络供应商的集群更改 MTU。

9.1. 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

您只能针对 OVN-Kubernetes 和 OpenShift SDN 集群网络供应商更改集群 MTU。

9.1.1. 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。

9.1.2. 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。具体开销由集群网络供应商决定：
 - OVN-Kubernetes: 100 字节
 - OpenShift SDN: 50 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所使用的最低 MTU 值中减去集群网络供应商的开销值。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**。

9.1.3. 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 9.1. 集群 MTU 的实时迁移

用户发起的步骤	OpenShift Container Platform 活动
<p>在 Cluster Network Operator 配置中设置以下值：</p> <ul style="list-style-type: none"> ● spec.migration.mtu.machine.to ● spec.migration.mtu.network.from ● spec.migration.mtu.network.to 	<p>Cluster Network Operator(CNO)：确认每个字段都设置为有效的值。</p> <ul style="list-style-type: none"> ● 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，以允许集群网络供应商的覆盖开销。对于 OVN-Kubernetes，开销为 100 字节，OpenShift SDN 的开销为 50 字节。 <p>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。</p> <p>Machine Config Operator(MCO)：执行集群中每个节点的滚动重启。</p>
<p>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：</p> <ul style="list-style-type: none"> ● 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 	<p>不适用</p>
<p>在集群网络供应商的 CNO 配置中设置 mtu 值，并将 spec.migration 设置为 null。</p>	<p>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。</p>

9.2. 更改集群 MTU

作为集群管理员，您可以更改集群的最大传输单元(MTU)。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。

以下流程描述了如何使用机器配置、DHCP 或 ISO 更改集群 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**)。
- 使用具有 **cluster-admin** 权限的用户登陆到集群。

- 已为集群识别目标 MTU。正确的 MTU 因集群使用的集群网络供应商而异：
 - OVN-Kubernetes: 集群 MTU 必须设置为比集群中的最低硬件 MTU 值小 100。
 - OpenShift SDN : 集群 MTU 必须设置为比集群中的最低硬件 MTU 值小 50。

流程

要增加或减少集群网络的 MTU，请完成以下步骤。

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:  OpenShiftSDN
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 集群网络供应商，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 集群网络供应商，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: 4.10.0
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: 4.10.0
```

```

metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf ❶
      contents:
        local: <interface>-mtu.conf ❷
        mode: 0600

```

- ❶ 指定主网络接口的 NetworkManager 连接名称。
- ❷ 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

```

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>**，对于 OVN-Kubernetes，值必须小 **100**，OpenShift SDN 必须小 **50**。

<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 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。

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
```

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/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 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。

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。

如果成功部署机器配置，则前面的输出包含 **/etc/NetworkManager/system-connections/<connection_name>** 文件路径。

机器配置不得包含 **ExecStart=/usr/local/bin/mtu-migration.sh** 行。

9. 要完成 MTU 迁移，请输入以下命令之一：

- 如果使用 OVN-Kubernetes 集群网络供应商：

```
$ 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。

- 如果使用 OpenShift SDN 集群网络供应商：

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": {"migration": null, "defaultNetwork":{"openshiftSDNConfig": {"mtu": <mtu> }}}}'
```

其中：

<mtu>

指定您使用 **<overlay_to>** 指定的新集群网络 MTU。

验证

您可以验证集群中的节点是否使用上一步中指定的 MTU。

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
```

9.3. 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 10 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。

10.1. 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。

10.2. 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"]
```

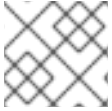
10.3. 其他资源

- [使用 NodePort 配置集群入口流量](#)
- [Network \[config.openshift.io/v1\]](#)
- [Service \[core/v1\]](#)

第 11 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 (IP failover) 在一组节点上管理一个虚拟 IP (VIP) 地址池。集合中的每个 VIP 都由从集合中选择的节点提供服务。只要单个节点可用, 就会提供 VIP。无法将 VIP 显式分发到节点上, 因此可能不存在没有 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。



注意

VIP 必须可以从集群外部路由。

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。

IP 故障转移使用 [Keepalived](#) 在一组主机上托管一组外部访问的 VIP 地址。在一个时间点上, 每个 VIP 仅由一个主机提供服务。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。

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。

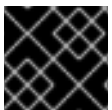
集群管理员可以通过 `OPENSSHIFT_HA_NOTIFY_SCRIPT` 变量提供一个脚本, 每当节点上的 VIP 的状态发生变化时会调用此脚本。keepalived 在为 VIP 提供服务时为 **master** 状态; 当另一个节点提供 VIP 服务时, 状态为 **backup**; 当检查脚本失败时, 状态为 **fault**。每当状态更改时, notify 脚本都会被调用, 并显示新的状态。

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, 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。这可以让任何 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。

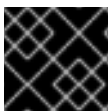
在使用 VIP 访问服务时, 任何节点都可以位于节点的 IP 故障转移集中, 因为无论应用容器集在哪里运行, 该服务都可以在所有节点上访问。任何 IP 故障转移节点可以随时变成主节点。服务可以使用外部 IP 和服务端口, 或者可以使用 **NodePort**。

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

设置 **NodePort** 是一个特权操作。



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 **ingressIP** 时，您可以将 IP 故障切换设置为与 **ingressIP** 范围相同的 VIP 范围。您还可以禁用监控端口。在本例中，所有 VIP 都出现在集群的同一节点上。任何用户都可以使用 **ingressIP** 设置服务，并使其具有高可用性。



重要

集群中最多有 254 个 VIP。

11.1. 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 11.1. IP 故障转移环境变量

变量名称	default	描述
OPENSIFT_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。
OPENSIFT_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 。
OPENSIFT_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。
OPENSIFT_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 。
OPENSIFT_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 。
OPENSIFT_HA_VIP_GROUPS		为 VRRP 创建的组数量。如果没有设置，则会为通过 OPENSIFT_HA_VIP_GROUPS 变量指定的每个虚拟 IP 范围创建一个组。
OPENSIFT_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。
OPENSIFT_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。
OPENSIFT_HA_CHECK_INTERVAL	2	检查脚本运行的期间（以秒为单位）。
OPENSIFT_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。

变量名称	default	描述
OPENSIFT_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。

11.2. 配置 IP 故障转移

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: OPENSIFT_HA_CONFIG_NAME
            value: "ipfailover"
          - name: OPENSIFT_HA_VIRTUAL_IPS 2
            value: "1.1.1.1-2"
          - name: OPENSIFT_HA_VIP_GROUPS 3
            value: "10"
          - name: OPENSIFT_HA_NETWORK_INTERFACE 4
            value: "ens3" #The host interface to assign the VIPs
          - name: OPENSIFT_HA_MONITOR_PORT 5
            value: "30060"
          - name: OPENSIFT_HA_VRRP_ID_OFFSET 6
            value: "0"
          - name: OPENSIFT_HA_REPLICA_COUNT 7
            value: "2" #Must match the number of replicas in the deployment
          - name: OPENSIFT_HA_USE_UNICAST
            value: "false"
          #- name: OPENSIFT_HA_UNICAST_PEERS
            #value: "10.0.148.40,10.0.160.234,10.0.199.110"
          - name: OPENSIFT_HA_IPTABLES_CHAIN 8
            value: "INPUT"
          #- name: OPENSIFT_HA_NOTIFY_SCRIPT 9

```

```

# value: /etc/keepalive/mynotifyscript.sh
- name: OPENSIFT_HA_CHECK_SCRIPT 10
  value: "/etc/keepalive/mycheckscript.sh"
- name: OPENSIFT_HA_PREEMPTION 11
  value: "preempt_delay 300"
- name: OPENSIFT_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 **OPENSIFT_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。

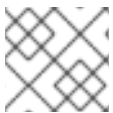
- 11 处理新的具有更高优先级主机的策略。默认值为 `preempt_delay 300`，这会导致，在有一个较低优先级的 master 提供 VIP 时，Keepalived 实例在 5 分钟后会接管 VIP。
- 12 检查脚本运行的期间（以秒为单位）。默认值为 `2`。
- 13 在创建部署之前创建 pull secret，否则您将在创建部署时收到错误。

11.3. 关于虚拟 IP 地址

keepalived 管理一组虚拟 IP 地址（VIP）。管理员必须确保所有这些地址：

- 可在集群外部配置的主机上访问。
- 不可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 最终都可能由不同的节点提供。

11.4. 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。

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）。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。

当处于 **master** 状态的节点上的 VIP 的检查脚本失败时，该节点上的 VIP 将进入 **fault** 状态，这会触发重新协商。在重新协商过程中，节点上没有处于 **fault** 状态的所有 VIP 都参与决定哪个节点接管 VIP。最后，VIP 在某些节点上进入 **master** 状态，VIP 则在其他节点上保持 **backup** 状态。

当具有 **backup** 状态的 VIP 的节点失败时，该节点上的 VIP 将进入 **fault** 状态。当检查脚本再次通过对 **fault** 状态的节点上的 VIP 检查时，该节点上的 VIP 将退出 **fault** 状态，并协商来进入 **master** 状态。然后，该节点上的 VIP 可能会进入 **master** 或 **backup** 状态。

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。keepalived 将以下三个参数传递给脚本：

- `$1` - `group` 或 `instance`
- `$2` - `group` 或 `instance` 的名称
- `$3` - 新状态：`master`、`backup` 或 `fault`

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配置映射。

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。脚本可以通过配置映射添加到 pod，如下所示。

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一个配置映射来容纳它。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
```

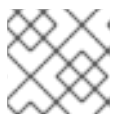
2. 创建配置映射：

```
$ oc create configmap mycustomcheck --from-file=mycheckscript.sh
```

3. 将脚本添加到容器集。挂载的配置映射文件的 **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 \
  OPENSIFT_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: OPENSIFT_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 **OPENSIFT_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**。

11.5. 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。但是，如果脱离 **fault** 状态的节点上的 VIP 具有更高的优先级，则抢占策略会决定其在集群中的角色。

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。

先决条件

- 已安装 OpenShift CLI (**oc**)。

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: OPENSIFT_HA_PREEMPTION ❶
      value: preempt_delay 300
  ...
```

❶ 设置 OPENSIFT_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。

11.6. 关于 VRRP ID 偏移

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每个 VIP，IP 故障转移 pod 必须分配对应的 **vrrp-id**。这可以从 **OPENSIFT_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 **OPENSIFT_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。

11.7. 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 **OPENSIFT_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。

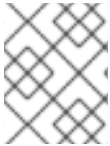
流程

- 要更改分配给每个组的 IP 地址数量，请更改 **OPENSIFT_HA_VIP_GROUPS** 变量的值，例如：

IP 故障转移配置的 Deployment YAML 示例

```
...
spec:
  env:
    - name: OPENSIFT_HA_VIP_GROUPS ❶
      value: "3"
...
```

- ❶ 如果在有七个 VIP 的环境中将 **OPENSIFT_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 **OPENSIFT_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。

11.8. INGRESSIP 的高可用性

在非云集群中，可以将 IP 故障转移和 **ingressIP** 合并到服务。其结果是，为使用 **ingressIP** 创建服务的用户提供了高可用性服务。

方法是指定一个 **ingressIPNetworkCIDR** 范围，然后在创建 **ipfailover** 配置时使用相同的范围。

由于 IP 故障转移最多可支持整个集群的 255 个 VIP，所以 **ingressIPNetworkCIDR** 需要为 **/24** 或更小。

11.9. 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:4.10
        command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
```

```
nodeSelector:  
  kubernetes.io/hostname: <host_name> <.>  
restartPolicy: Never
```

<.> 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 12 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)。

12.1. 支持 OPENSIFT 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。

12.1.1. 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

```

12.2. 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
```

12.3. 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/ubi8/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/ubi8/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 **sctpserver** 服务的 IP 地址。使用以下命令：

```
$ oc get services sctpsservice -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 **sctpsservice** 服务的集群 IP 地址。

```
# nc <cluster_IP> 30102 --sctp
```

第 13 章 使用 PTP 硬件



重要

使用配置为边界时钟的单一 NIC 的精确时间协议(PTP)硬件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](#)。

13.1. 关于 PTP 硬件

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。



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。

您可以通过部署 PTP Operator，使用 OpenShift Container Platform 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。

13.2. 关于 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。

linuxptp 软件包包括用于时钟同步的 **ptp4l** 和 **phc2sys** 程序。**ptp4l** 实现 PTP 边界时钟和普通时钟。**ptp4l** 将 PTP 硬件时钟与硬件时间戳同步，并将系统时钟与源时钟与软件时间戳同步。**phc2sys** 用于硬件时间戳，将系统时钟与网络接口控制器 (NIC) 上的 PTP 硬件时钟同步。

13.2.1. 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 同步时钟以源目标层次结构进行组织。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。目标时钟与源时钟同步，目标时钟本身也可以是其他下游时钟的源。以下时钟类型可以包含在配置中：

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 时钟可同步到全球位置系统(GPS)时间源。

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。

Boundary 时钟

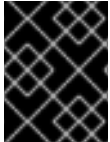
Boundary (边界) 时钟在两个或更多个通信路径中具有端口，并且可以是指向其他目标时钟的源和目

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。

13.2.2. 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](#)。

13.3. 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-ntp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ntp
    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
4.10.0-202201261535                Succeeded

```

13.4. 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。

13.5. 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。

13.5.1. 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。

13.5.2. 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。

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** 服务配置为特定硬件和环境的 grandmaster 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

- 在裸机集群主机上安装 Intel Westport Channel 网络接口。
- 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** 资源。例如：
 - a. 将以下 YAML 保存到 **grandmaster-clock-ptp-config.yaml** 文件中：

PTP grandmaster 时钟配置示例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster-clock
  namespace: openshift-ptp
  annotations: {}
```

```
spec:
  profile:
    - name: grandmaster-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        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: grandmaster-clock
  priority: 4
  match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"

```

- b. 运行以下命令来创建 CR :

```
$ oc create -f grandmaster-clock-ntp-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-ntp** 命名空间中的 pod 列表：

```
$ oc get pods -n openshift-ntp -o wide
```

输出示例

```

NAME                                READY STATUS RESTARTS AGE IP             NODE
linuxntp-daemon-74m2g              3/3   Running 3      4d15h 10.16.230.7   compute-1.example.com
ntp-operator-5f4f48d7c-x7zkg      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 **linuxntp** 守护进程的日志。运行以下命令：

```
$ oc logs linuxntp-daemon-74m2g -n openshift-ntp -c linuxntp-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

```

13.5.3. 将 **linuxntp** 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 **linuxntp** 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 **ptp4IOpts**、**ptp4IConf** 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
      ptp4IOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4IConf: |
        [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 13.1. PTP 普通时钟 CR 配置选项

自定义资源字段	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。

自定义资源字段	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。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	可选的。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	对于普通时钟，将 .recommend.priority 设置为 0 。
.recommend.match	使用 nodeLabel 或 nodeName 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 更新 nodeLabel 。例如： node-role.kubernetes.io/worker 。

自定义资源字段	描述
<code>.recommend.match.nodeLabel</code>	通过 <code>oc get nodes</code> 命令，使用来自节点对象的 <code>node.Name</code> 值更新 <code>nodeName</code> 。例如： <code>compute-0.example.com</code> 。

- 运行以下命令来创建 **PtpConfig** CR：

```
$ oc create -f ordinary-clock-ntp-config.yaml
```

验证

- 检查 **PtpConfig** 配置集是否已应用到节点。
 - 运行以下命令，获取 **openshift-ntp** 命名空间中的 pod 列表：

```
$ oc get pods -n openshift-ntp -o wide
```

输出示例

```
NAME                READY STATUS RESTARTS AGE IP           NODE
linuxntp-daemon-4xkbb 1/1   Running 0       43m 10.1.196.24 compute-0.example.com
linuxntp-daemon-tdspf 1/1   Running 0       43m 10.1.196.25 compute-1.example.com
ntp-operator-657bbb64c8-2f8sj 1/1   Running 0       43m 10.129.0.61 control-plane-1.example.com
```

- 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 **linuxntp** 守护进程的日志。运行以下命令：

```
$ oc logs linuxntp-daemon-4xkbb -n openshift-ntp -c linuxntp-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] Ptp4IOpts: -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

- 如需有关 PTP 硬件上的 FIFO 优先级调度的更多信息，请参阅[PTP 硬件配置 FIFO 优先级调度](#)。
- 有关配置 PTP 快速事件的更多信息，请参阅[配置 PTP 快速事件通知发布程序](#)。

13.5.4. 将 linuxntp 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 13.2. PTP 边界时钟 CR 配置选项

自定义资源字段	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。

自定义资源字段	描述
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。
ptpClockThreshold	可选的。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。

自定义资源字段	描述
<code>.recommend.priority</code>	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。
<code>.recommend.match</code>	使用 nodeLabel 或 nodeName 指定 .recommend.match 规则。
<code>.recommend.match.nodeLabel</code>	通过 <code>oc get nodes --show-labels</code> 命令，使用来自节点对象的 node.Labels 的 key 更新 nodeLabel 。例如： <code>node-role.kubernetes.io/worker</code> 。
<code>.recommend.match.nodeName</code>	通过 <code>oc get nodes</code> 命令，使用来自节点对象的 node.Name 值更新 nodeName 。例如： <code>compute-0.example.com</code> 。

- 运行以下命令来创建 CR：

```
$ oc create -f boundary-clock-ntp-config.yaml
```

验证

- 检查 **PtpConfig** 配置集是否已应用到节点。
 - 运行以下命令，获取 **openshift-ntp** 命名空间中的 pod 列表：

```
$ oc get pods -n openshift-ntp -o wide
```

输出示例

```
NAME                READY STATUS RESTARTS AGE IP           NODE
linuxntp-daemon-4xkbb 1/1   Running 0       43m 10.1.196.24  compute-0.example.com
linuxntp-daemon-tdspf 1/1   Running 0       43m 10.1.196.25  compute-1.example.com
ntp-operator-657bbb64c8-2f8sj 1/1   Running 0       43m 10.129.0.61  control-plane-1.example.com
```

- 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 **linuxntp** 守护进程的日志。运行以下命令：

```
$ oc logs linuxntp-daemon-4xkbb -n openshift-ntp -c linuxntp-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

- 如需有关 PTP 硬件上的 FIFO 优先级调度的更多信息，请参阅[PTP 硬件配置 FIFO 优先级调度](#)。
- 有关配置 PTP 快速事件的更多信息，请参阅[配置 PTP 快速事件通知发布程序](#)。

13.5.5. 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 配置进行的更改，以便使用 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 13.3. 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](#)。

13.5.6. 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配置中，PTP 守护进程线程在受限制的 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-ntp
```

2. 更改 **ptpSchedulingPolicy** 和 **ptpSchedulingPriority** 字段：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ntp
...
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-ntp -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-ntp 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
```

13.6. 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-ntp -o wide
```

输出示例

```
NAME                                READY STATUS RESTARTS AGE IP      NODE
linuxntp-daemon-lmvgn              3/3   Running 0       4d17h 10.1.196.24 compute-0.example.com
linuxntp-daemon-qhfg7              3/3   Running 0       4d17h 10.1.196.25 compute-1.example.com
ntp-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 **linuxntp-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-ntp get nodeptpdevices.ntp.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-ntp get nodeptpdevices.ntp.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

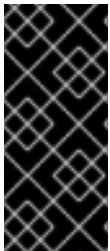
■

```

sending: GET_PORT_DATA_SET
40a6b7.ffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      40a6b7.ffe.166ef0-1
portState        SLAVE
logMinDelayReqInterval -4
peerMeanPathDelay 0
logAnnounceInterval -3
announceReceiptTimeout 3
logSyncInterval -4
delayMechanism 1
logMinPdelayReqInterval -4
versionNumber 2

```

13.7. PTP 硬件快速事件通知框架



重要

带有普通时钟的 PTP 事件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](#)。

13.7.1. 关于 PTP 和时钟同步错误事件

虚拟 RAN 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。快速事件通知是有关即将到来和实时安全时间协议 (PTP) 时钟同步事件的早期警告信号。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。

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 RAN 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。

OpenShift Container Platform 中的 PTP Operator 为支持 PTP 的每个网络接口生成快速事件通知。通过高级消息队列协议 (AMQP) 消息总线，可使用 **cloud-event-proxy** sidecar 容器提供这些事件。AMQP 消息总线由 AMQ Interconnect Operator 提供。



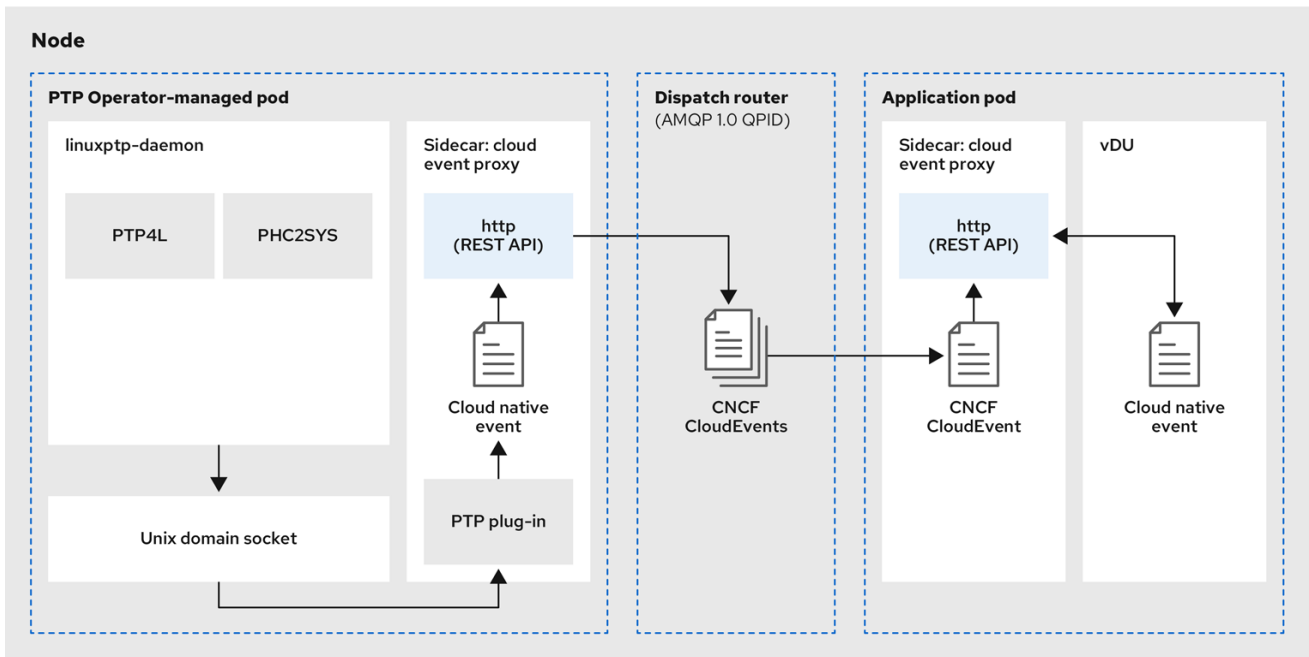
注意

PTP 快速事件通知可用于配置为使用 PTP 普通时钟或 PTP 边界时钟。

13.7.2. 关于 PTP 快速事件通知框架

您可以将分布式单元 (DU) 应用程序订阅到 Precision Time Protocol (PTP) 快速事件通知，这些通知由 OpenShift Container Platform 使用 PTP Operator 和 **cloud-event-proxy** sidecar 容器生成。您可以通过在 **ptpOperatorConfig** 自定义资源 (CR) 中将 **enableEventPublisher** 字段设置为 **true** 来启用 **cloud-event-proxy** sidecar 容器，并指定 Advanced Message Queue Protocol (AMQP) **transportHost** 地址。PTP fast 事件使用 AMQ Interconnect Operator 提供的 AMQP 事件通知总线。AMQ Interconnect 是 Red Hat AMQ 的一个组件，它是在支持 AMQP 的端点之间提供灵活的消息路由的消息传递路由器。以下是 PTP 快速事件框架的概述：

图 13.1. PTP 快速事件概述



218_OpenShift_0122

cloud-event-proxy sidecar 容器可以在不使用主应用程序的任何资源的情况下访问与主 vRAN 应用程序相同的资源，且无显著延迟。

快速事件通知框架使用 REST API 进行通信，并且基于 O-RAN REST API 规范。框架由发布者、订户和 AMQ 消息传递总线组成，以处理发布者和订阅者应用程序之间的通信。**cloud-event-proxy** sidecar 是一个在 pod 中运行的实用程序容器，它与 DU 节点上的主 DU 应用程序容器松散耦合。它提供了一个事件发布框架，允许您订阅 DU 应用程序来发布的 PTP 事件。

DU 应用程序以 sidecar 模式运行 **cloud-event-proxy** 容器，以订阅 PTP 事件。以下工作流描述了 DU 应用程序如何使用 PTP 快速事件：

1. **DU 应用程序请求一个订阅**：DU 将 API 请求发送到 **cloud-event-proxy** sidecar 以创建 PTP 事件订阅。**cloud-event-proxy** sidecar 创建一个订阅资源。
2. **cloud-event-proxy sidecar 创建订阅**：事件资源由 **cloud-event-proxy** sidecar 保留。**cloud-event-proxy** sidecar 容器发送带有 ID 和 URL 位置的确认，以访问存储的订阅资源。sidecar 为订阅中指定的资源创建一个 AMQ 消息传递监听程序协议。
3. **DU 应用程序接收 PTP 事件通知**：**cloud-event-proxy** sidecar 容器侦听资源限定器中指定的地址。DU 事件消费者处理消息并将其传递到订阅中指定的返回 URL。
4. **cloud-event-proxy sidecar 验证 PTP 事件并将其发布到 DU 应用程序**：**cloud-event-proxy** sidecar 接收事件，解封云事件对象以检索数据，并获取返回 URL 以将事件发回到 DU 消费者应用程序。
5. **DU 应用程序使用 PTP 事件**：DU 应用程序事件消费者接收和处理 PTP 事件。

13.7.3. 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。您可以通过安装 AMQ Interconnect Operator 来在集群中使用。

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。

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
```

13.7.4. 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

- 安装 OpenShift Container Platform CLI (**oc**)。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator 和 AMQ Interconnect 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
    transportHost: amqp://<instance_name>.<namespace>.svc.cluster.local ②

```

- ① 将 **enableEventPublisher** 设置为 **true** 以启用 PTP 快速事件通知。
- ② 将 **transportHost** 设置为您配置的 AMQ 路由器，其中 **<instance_name>** 和 **<namespace>** 对应于 AMQ Interconnect 路由器实例名称和命名空间，如 **amqp://amq-interconnect.amq-interconnect.svc.cluster.local**

b. 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 **ptp4IOpts** 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"
      ptp4IOpts: "-2 -s --summary_interval -4" ①
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ②
      ptp4IConf: "" 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 指定包含配置来替换默认 **/etc/ptp4l.conf** 文件的字符串。要使用默认配置，请将字段留空。
- ④ 可选的。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](#)。

13.7.5. 将 DU 应用程序订阅到 PTP 事件 REST API 参考

使用 PTP 事件通知 REST API 将分布式单元(DU)应用程序订阅到父节点上生成的 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/subscriptions/status/<subscription_id>`
 - **PUT** : 为指定订阅 ID 创建新状态 ping 请求
- `/api/ocloudNotifications/v1/health`
 - **GET** : 返回 **ocloudNotifications** API 的健康状况



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。

13.7.5.1. api/ocloudNotifications/v1/subscriptions

13.7.5.1.1. HTTP 方法

GET api/ocloudNotifications/v1/subscriptions

13.7.5.1.1.1. 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"
  }
]

```

13.7.5.1.2. HTTP 方法

POST api/ocloudNotifications/v1/subscriptions

13.7.5.1.2.1. 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 13.4. 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"
}

```

13.7.5.2. api/ocloudNotifications/v1/subscriptions/<subscription_id>

13.7.5.2.1. HTTP 方法

GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

13.7.5.2.1.1. 描述

返回 ID 为 <subscription_id> 的订阅详情

表 13.5. 查询参数

参数	类型
<subscription_id>	字符串

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"
}
```

13.7.5.3. api/ocloudNotifications/v1/subscriptions/status/<subscription_id>

13.7.5.3.1. HTTP 方法

PUT api/ocloudNotifications/v1/subscriptions/status/<subscription_id>

13.7.5.3.1.1. 描述

为 ID 为 <subscription_id> 的订阅创建一个新的状态 ping 请求。如果订阅存在，状态请求将成功，并返回 **202 Accepted** 状态代码。

表 13.6. 查询参数

参数	类型
<subscription_id>	字符串

API 响应示例

```
{"status":"ping sent"}
```

13.7.5.4. api/ocloudNotifications/v1/health/

13.7.5.4.1. HTTP 方法

GET api/ocloudNotifications/v1/health/

13.7.5.4.1.1. 描述

返回 **ocloudNotifications** REST API 的健康状况。

API 响应示例

```
OK
```

13.7.6. 使用 CLI 监控 PTP 快速事件指标

您可以使用 **oc** CLI 直接从 **cloud-event-proxy** 容器监控快速事件总线指标。



注意

OpenShift Container Platform Web 控制台中还提供了 PTP fast 事件通知指标。

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。
- 安装和配置 PTP Operator。

流程

1. 获取活跃的 **linuxptp-daemon** pod 列表。

```
$ oc get pods -n openshift-ntp
```

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 **cloud-event-proxy** 容器的指标：

```
$ oc exec -it <linuxptp-daemon> -n openshift-ntp -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_amqp_events_published Metric to get number of events published by the transport
# TYPE cne_amqp_events_published gauge
cne_amqp_events_published{address="/cluster/node/compute-1.example.com/ntp/status",status="success"} 1041
# HELP cne_amqp_events_received Metric to get number of events received by the transport
# TYPE cne_amqp_events_received gauge
cne_amqp_events_received{address="/cluster/node/compute-1.example.com/ntp",status="success"} 1019
# HELP cne_amqp_receiver Metric to get number of receiver created
# TYPE cne_amqp_receiver gauge
cne_amqp_receiver{address="/cluster/node/mock",status="active"} 1
cne_amqp_receiver{address="/cluster/node/compute-1.example.com/ntp",status="active"} 1
cne_amqp_receiver{address="/cluster/node/compute-1.example.com/redfish/event",status="active"}
...
```

13.7.7. 在 web 控制台中监控 PTP fast 事件指标

您可以使用预配置和自我更新 Prometheus 监控堆栈在 OpenShift Container Platform Web 控制台中监控 PTP 快速事件指标。

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。

流程

1. 输入以下命令从 **cloud-event-proxy** sidecar 容器返回可用 PTP 指标列表：

```
$ oc exec -it <linuxptp_daemon_pod> -n openshift-ptp -c cloud-event-proxy -- curl 127.0.0.1:9091/metrics
```

其中：

<linuxptp_daemon_pod>

指定您要查询的 pod，例如 **linuxptp-daemon-2t78p**。

2. 从返回的指标列表中复制您要查询的 PTP 指标名称，例如 **cne_amqp_events_received**。
3. 在 OpenShift Container Platform web 控制台中点 **Observe → Metrics**。
4. 将 PTP 指标粘贴到 **Expression** 字段中，点 **Run query**。

其他资源

- [管理指标](#)

第 14 章 外部 DNS OPERATOR

14.1. OPENSIFT 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。

14.1.1. 外部 DNS Operator

External DNS Operator 从 **olm.openshift.io** API 组实现外部 DNS API。External DNS Operator 使用部署资源部署 **ExternalDNS**。ExternalDNS 部署会监视集群中服务和路由等资源，并更新外部 DNS 供应商。

流程

您可以根据 OperatorHub 的要求部署 ExternalDNS Operator，这会创建一个 **Subscription** 对象。

1. 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. 检查安装计划的状态，安装计划的状态必须为 **Complete**：

```
$ oc -n external-dns-operator get ip <install_plan_name> -o yaml | yq .status.phase'
```

输出示例

```
Complete
```

3. 使用 **oc get** 命令来查看部署状态：

```
$ 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

14.1.2. 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. 查看外部 DNS Operator 的日志：

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

14.2. 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。

14.2.1. 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-v1.0**。
 - 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**。

14.3. 外部 DNS OPERATOR 配置参数

外部 DNS Operator 包括以下配置参数：

14.3.1. 外部 DNS Operator 配置参数

External DNS Operator 包括以下配置参数：

参数	描述
spec	<p>启用云供应商的类型。</p> <pre>spec: provider: type: AWS 1 aws: credentials: name: aws-access-key 2</pre> <p>1 定义可用的选项，如 AWS、GCP 和 Azure。</p> <p>2 定义包含云供应商凭证的 secret 名称。</p>
zones	<p>允许您根据域指定 DNS 区域。如果没有指定区，ExternalDNS 会发现云供应商帐户中存在的所有区域。</p> <pre>zones: - "myzoneid" 1</pre> <p>1 指定 DNS 区 ID。</p>
domains	<p>允许您根据域指定 AWS 区域。如果没有指定域，ExternalDNS 会发现云供应商帐户中存在所有区域。</p> <pre>domains: - filterType: Include 1 matchType: Exact 2 name: "myzonedomain1.com" 3 - filterType: Include matchType: Pattern 4 pattern: ".*\\.otherzonedomain\\.com" 5</pre> <p>1 指示 ExternalDNS 包含指定的域。</p> <p>2 指示 ExternalDNS 指示域匹配必须完全匹配，而不是正则表达式匹配。</p> <p>3 定义 ExternalDNS 过滤器的确切域名。</p> <p>4 在 ExternalDNS 中设置 regex-domain-filter 标志。您可以使用 Regex 过滤器来限制可能的域。</p> <p>5 定义 ExternalDNS 使用的 regex 模式来过滤目标区域的域。</p>

参数	描述
source	<p>允许您指定 DNS 记录、Service 或 Route 的源。</p> <pre> 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 </pre> <p>1 定义 DNS 记录源的设置。</p> <p>2 ExternalDNS 使用 Service 类型作为创建 dns 记录的源。</p> <p>3 在 ExternalDNS 中设置 service-type-filter 标志。serviceType 包含以下字段：</p> <ul style="list-style-type: none"> ● 默认: LoadBalancer ● 预期: ClusterIP ● NodePort ● LoadBalancer ● ExternalName <p>4 确保控制器只考虑与标签过滤器匹配的资源。</p> <p>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。</p> <p>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。</p> <pre> source: type: OpenShiftRoute 1 openshiftRouteOptions: routerName: default 2 labelFilter: matchLabels: external-dns.mydomain.org/publish: "yes" </pre> <p>1 ExternalDNS 使用类型 route 作为创建 dns 记录的源。</p> <p>2 如果源是 OpenShiftRoute，您可以传递 Ingress Controller 名称。ExternalDNS 使用 Ingress Controller 的规范名称作为 CNAME 记录的目标。</p>

参数	描述
----	----

14.4. 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。

14.4.1. 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/v1alpha1
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
```

14.5. 在 AZURE 上创建 DNS 记录

您可以使用 External DNS Operator 在 Azure 上创建 DNS 记录。

14.5.1. 使用 Red Hat External DNS Operator 在 Azure 公共 DNS 区上创建 DNS 记录

您可以使用 Red Hat External DNS Operator 为 Azure 的公共 DNS 区域创建 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** 命名空间中获取 azure-credentials secret 的值。

```
$ 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)
```

3. 使用 base64 解码值登录到 azure：

```
$ az login --service-principal -u "${CLIENT_ID}" -p "${CLIENT_SECRET}" --tenant "${TENANT_ID}"
```

4. 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.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
```

5. 获取 dns zones 列表以查找与之前找到的路由域对应的 dns 区域：

```
$ az network dns zone list --resource-group "${RESOURCE_GROUP}"
```

6. 为路由源创建 ExternalDNS 资源：

```
apiVersion: externaldns.olm.openshift.io/v1alpha1
kind: ExternalDNS
metadata:
  name: sample-azure 1
spec:
  zones:
    - "/subscriptions/1234567890/resourceGroups/test-azure-xxxx-
```

```

rg/providers/Microsoft.Network/dnszones/test.azure.example.com" 2
provider:
  type: Azure 3
source:
  openshiftRouteOptions: 4
    routerName: default 5
    type: OpenShiftRoute 6
EOF

```

- 1 指定外部 DNS CR 的名称。
- 2 定义区域 ID。
- 3 定义 Azure DNS 供应商。
- 4 您可以定义 DNS 记录源的选项。
- 5 如果源是 **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 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。

7. 使用以下命令，检查为 OCP 路由创建的记录：

```
$ az network dns record-set list -g "${RESOURCE_GROUP}" -z test.azure.example.com |
grep console
```



注意

要在私有 Azure dns 上的私有托管区中创建记录，您需要在 **zones** 下指定私有区，在 **ExternalDNS** 容器 args 中填充供应商类型到 **azure-private-dns**。

14.6. 在 GCP 上创建 DNS 记录

您可以使用 External DNS Operator 在 GCP 上创建 DNS 记录。

14.6.1. 使用 Red Hat External DNS Operator 在 GCP 公共管理区上创建 DNS 记录

您可以使用 Red Hat External DNS Operator 在公共受管区上为 GCP 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
```

- 运行以下命令，将 gcp-credentials secret 中的 service_account.json 值复制到编码-gcloud.json 的文件中：

```
$ oc get secret gcp-credentials -n kube-system --template='{{$v := index .data "service_account.json"}}{{$v}}' | base64 -d - > decoded-gcloud.json
```

- 导出 Google 凭证：

```
$ export GOOGLE_CREDENTIALS=decoded-gcloud.json
```

- 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
```

- 设置项目：

```
$ gcloud config set project <project_id as per decoded-gcloud.json>
```

- 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.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
```

- 获取受管区列表以查找与之前找到的路由域对应的区：

```
$ gcloud dns managed-zones list | grep test.gcp.example.com
qe-cvs4g-private-zone test.gcp.example.com
```

- 为路由源创建 ExternalDNS 资源：

```
apiVersion: externaldns.olm.openshift.io/v1alpha1
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
EOF
```

- 1 指定外部 DNS CR 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 Google Cloud DNS 供应商。
- 6 您可以定义 DNS 记录源的选项。
- 7 如果源是 **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 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。

9. 使用以下命令，检查为 OCP 路由创建的记录：

```
$ gcloud dns record-sets list --zone=qe-cvs4g-private-zone | grep console
```

14.7. 在外部 DNS OPERATOR 上配置集群范围代理

您可以在 External DNS Operator 中配置集群范围代理。在外部 DNS Operator 中配置集群范围代理后，Operator Lifecycle Manager (OLM) 会自动使用环境变量（如 **HTTP_PROXY**、**HTTPS_PROXY**）和 **NO_PROXY** 等环境变量更新 Operator 的所有部署。

14.7.1. 配置外部 DNS Operator 以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 15 章 网络策略

15.1. 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。

15.1.1. 关于网络策略

在使用支持 Kubernetes 网络策略的 Kubernetes Container Network Interface (CNI) 插件的集群中，网络隔离完全由 **NetworkPolicy** 对象控制。

在 OpenShift Container Platform 4.10 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。

OpenShift SDN 集群网络供应商现在支持出口网络策略，如 **egress** 字段指定。



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。

网络策略只适用于 TCP、UDP 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。

15.1.2. 网络策略优化

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。



注意

有效使用网络策略规则的指南只适用于 OpenShift SDN 集群网络供应商。

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。

15.1.3. 后续步骤

- [创建网络策略](#)
- 可选：[定义默认网络策略](#)

15.1.4. 其他资源

- [项目和命名空间](#)
- [配置多租户网络策略](#)
- [网络策略 API](#)

15.2. 记录网络策略事件

作为集群管理员，您可以为集群配置网络策略审计日志记录，并为一个或多个命名空间启用日志记录。



注意

网络策略的审计日志记录仅适用于 [OVN-Kubernetes 集群网络供应商](#)。

15.2.1. 网络策略审计日志记录

OVN-Kubernetes 集群网络供应商使用 Open Virtual Network (OVN) ACL 管理网络策略。审计日志记录会公开允许和拒绝 ACL 事件。

您可以为网络策略审计日志（如 syslog 服务器或 UNIX 域套接字）配置目的地。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`。

网络策略审计日志记录通过 [k8s.ovn.org/acl-logging](#) 键注解命名空间来启用每个命名空间，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：

ACL 拒绝日志条目示例

```
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all",
verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=
10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

下表描述了命名空间注解值：

表 15.1. 网络策略审计日志记录命名空间注解

注解	值
k8s.ovn.org/acl-logging	<p>您必须至少指定 allow、deny 或同时指定两者才能为命名空间启用网络策略审计日志记录。</p> <p>deny 可选：指定 alert、warning、notice、info 或 debug。</p> <p>allow 可选：指定 alert、warning、notice、info 或 debug。</p>

15.2.2. 网络策略审计配置

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 演示了网络策略审计日志记录功能的默认值。

审计日志记录配置

```

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

```

下表描述了网络策略审计日志记录的配置字段。

表 15.2. policyAuditConfig 对象

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。

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

15.2.3. 为集群配置网络策略审计

作为集群管理员，您可以自定义集群的网络策略审计日志记录。

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

- 要自定义网络策略审计日志记录配置，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. 启用审计日志记录：

```
$ oc annotate namespace verify-audit-logging k8s.ovn.org/acl-logging='{ "deny": "alert",
"allow": "alert" }'
```

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

c. 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
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector: {}
  egress:
  - to:
```

```

- namespaceSelector:
  matchLabels:
    namespace: verify-audit-logging
EOF

```

输出示例

```

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

```

- 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

```

- 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

```

- 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 :

- 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 ping **POD_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

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of the containers in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
```

```
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

15.2.4. 为命名空间启用网络策略审计日志记录

作为集群管理员，您可以为命名空间启用网络策略审计日志记录。

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

- 要为命名空间启用网络策略审计日志记录，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

```
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all", verdict=drop, severity=alert: icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

15.2.5. 禁用命名空间的网络策略审计日志记录

作为集群管理员，您可以为命名空间禁用网络策略审计日志记录。

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

- 要禁用命名空间的网络策略审计日志记录，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
```

15.2.6. 其他资源

- [关于网络策略](#)

15.3. 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。

15.3.1. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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

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

```

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/default-deny 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。

15.3.2. 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。

15.3.3. 其他资源

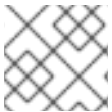
- [访问Web控制台](#)

15.4. 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。

15.4.1. 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。

15.4.2. 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。

15.5. 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。

15.5.1. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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 控制台策略编辑网络策略。

15.5.2. 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。

15.5.3. 其他资源

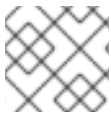
- [创建网络策略](#)

15.6. 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。

15.6.1. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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。

15.7. 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。

15.7.1. 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：

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。

15.7.2. 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 网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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** 权限的用户登录到集群。
- 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> ❶
```

❶ 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
```

15.8. 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 集群网络供应商，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，但具有设置网络策略模式。

15.8.1. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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
```

15.8.2. 后续步骤

- [定义默认网络策略](#)

15.8.3. 其他资源

- [OpenShift SDN 网络隔离模式](#)

第 16 章 多网络

16.1. 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。

16.1.1. 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。

16.1.2. 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。
- **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。
- **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。

16.2. 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](#)
- [IPVLAN](#)
- [MACVLAN](#)

16.2.1. 管理额外网络的方法

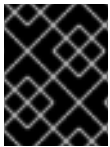
您可以通过两种方法来管理额外网络的生命周期。每种方法都是相互排斥的，您一次只能使用一种方法来管理额外网络。对于任一方法，额外网络由您配置的 Container Network Interface(CNI)插件管理。

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。

16.2.2. 配置额外网络附加

额外网络通过 **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 16.1. NetworkAttachmentDefinition API 字段

字段	类型	描述
metadata.name	字符串	额外网络的名称。
metadata.namespace	字符串	与对象关联的命名空间。
spec.config	字符串	JSON 格式的 CNI 插件配置。

16.2.2.1. 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: ❶
  - name: <name> ❷
    namespace: <namespace> ❸
    rawCNIConfig: |- ❹
      {
        ...
      }
  type: Raw

```

- ❶ 由一个或多个附加网络配置组成的数组。
- ❷ 您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。
- ❸ 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。
- ❹ JSON 格式的 CNI 插件配置。

16.2.2.2. 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> ❶
spec:
  config: |- ❷
    {
      ...
    }

```

- ❶ 您要创建的额外网络附加的名称。
- ❷ JSON 格式的 CNI 插件配置。

16.2.3. 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。

16.2.3.1. 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 16.2. bridge CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	字符串	CNI 规格版本。需要 0.3.1 值。
name	字符串	您之前为 CNO 配置提供的 name 参数的值。
type	字符串	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	字符串	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 <i>反射中继</i> 。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 。
vlan	字符串	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	字符串	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	字符串	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，您需要使用以下命令在 uplink 接口上允许 vlan：

```
$ bridge vlan add vid VLAN_ID dev DEV
```

16.2.3.1.1. 网桥配置示例

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"
  }
}
```

16.2.3.2. 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 16.3. 主机 device CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	字符串	CNI 规格版本。需要 0.3.1 值。
name	字符串	您之前为 CNO 配置提供的 name 参数的值。
type	字符串	用于配置的 CNI 插件的名称： host-device 。
device	字符串	可选：设备的名称，如 eth0 。
hwaddr	字符串	可选：设备硬件 MAC 地址。
kernelpath	字符串	可选：Linux 内核设备路径，如 /sys/devices/pci0000:00/0000:00:1f.6 。

字段	类型	描述
pciBusID	字符串	可选：网络设备的 PCI 地址，如 0000:00:1f.6 。

16.2.3.2.1. 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"
}
```

16.2.3.3. 配置 IPVLAN 额外网络

以下对象描述了 IPVLAN CNI 插件的配置参数：

表 16.4. IPVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	字符串	CNI 规格版本。需要 0.3.1 值。
name	字符串	您之前为 CNO 配置提供的 name 参数的值。
type	字符串	要配置的 CNI 插件的名称： ipvlan 。
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	字符串	可选：虚拟网络的操作模式。这个值必须是 I2 、 I3 或 I3s 。默认值为 I2 。
master	字符串	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。



注意

- **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。

16.2.3.3.1. ipvlan 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",
  "mode": "I3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

16.2.3.4. 配置 MACVLAN 额外网络

以下对象描述了 **macvlan** CNI 插件的配置参数：

表 16.5. MACVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	字符串	CNI 规格版本。需要 0.3.1 值。
name	字符串	您之前为 CNO 配置提供的 name 参数的值。
type	字符串	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	字符串	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 。

字段	类型	描述
master	字符串	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	字符串	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。



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。

16.2.3.4.1. macvlan 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",
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
  }
}
```

16.2.4. 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。

16.2.4.1. 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 16.6. ipam 静态配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。值必须是 static 。

字段	类型	描述
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 16.7. ipam.addresses[] array

字段	类型	描述
address	字符串	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	字符串	出口网络流量要路由到的默认网关。

表 16.8. ipam.routes[] array

字段	类型	描述
dst	字符串	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	字符串	网络流量路由的网关。

表 16.9. 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"
      }
    ]
  }
}

```

16.2.4.2. 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 16.10. ipam DHCP 配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。需要值 dhcp 。

动态 IP 地址(DHCP)分配配置示例

```

{

```



```

    "ipam": {
      "type": "dhcp"
    }
  }
}

```

16.2.4.3. 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。

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置：

表 16.11. ipam whereabouts 配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。需要 abouts 的值。
range	字符串	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。

使用 Whereabouts 的动态 IP 地址分配配置示例

```

{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}

```

16.2.4.4. 创建 Whereabouts 协调器守护进程集

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** 自定义资源进行动态 IP 地址分配。

通过 Cluster Network Operator 配置额外网络时，Whereabouts reconciler 守护进程集会被自动创建。从 YAML 清单配置额外网络时，它不会自动创建。

要触发 Whereabouts 协调器 daemonset 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 **whereabouts-shim** 网络附加。

使用以下步骤部署 Whereabouts 协调器 daemonset。

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. 修改 CR 中的 **additionalNetworks** 参数，以添加 **abouts-shim** 网络附加定义。例如 :

```
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-p4sxx 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
```

16.2.5. 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
```

16.2.6. 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。

16.3. 关于虚拟路由和转发

16.3.1. 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。

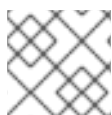
进程可将套接字绑定到 VRF 设备。通过绑定套接字的数据包使用与 VRF 设备关联的路由表。VRF 的一个重要特性是，它只影响 OSI 模型层 3 以上的流量，因此 L2 工具（如 LLDP）不会受到影响。这可使优先级更高的 IP 规则（如基于策略的路由）优先于针对特定流量的 VRF 设备规则。

16.3.1.1. 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。

16.4. 配置多网络策略

作为集群管理员，您可以为额外网络配置网络策略。



注意

您只能为 macvlan 额外网络指定多网络策略。不支持其他类型的额外网络，如 ipvlan。

16.4.1. 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 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。

16.4.2. 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
```

16.4.3. 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。

16.4.3.1. 先决条件

- 您已为集群启用了多网络策略支持。

16.4.3.2. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    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。

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>

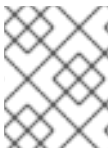
指定多网络策略文件名。

<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/default-deny 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。

16.4.3.3. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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。

16.4.3.4. 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。

16.4.3.5. 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** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。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。

16.4.4. 其他资源

- [关于网络策略](#)
- [了解多网络](#)
- [配置 macvlan 网络](#)

16.5. 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。

16.5.1. 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/networks-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/networks-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。

16.5.1.1. 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 **元数据** 映射中。**k8s.v1.cni.cncf.io/networks** 接受一个 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。

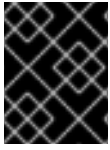
3. 在以下示例中，通过 **default-route** 参数，注解指定了哪个网络附加将使用默认路由。

```
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 route** 命令。

```
$ oc exec -it <pod_name> -- ip route
```



注意

您还可以引用 pod 的 **k8s.v1.cni.cncf.io/networks-status** 来查看哪个额外网络已经分配了默认路由，这可以通过 JSON 格式的对象列表中的 **default-route** 键实现。

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 `rawCNICongig` 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
rawCNICongig: '{ 3
  ...
}'
type: Raw
```

- 1 为您要创建的额外网络附加指定名称。名称在指定的命名空间中必须是唯一的。
- 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 }, ❸
    "master": "eth0", ❹
    "mode": "bridge",
    "ipam": {
      "type": "static"
    }
  }, {
    "capabilities": { "mac": true }, ❺
    "type": "tuning"
  }
}

```

- ❶ 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。
- ❷ 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。
- ❸ 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。
- ❹ 指定 macvlan 插件使用的接口。
- ❺ 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>", ❶
        "ips": [ "192.0.2.205/24" ], ❷
        "mac": "CA:FE:C0:FF:EE:00" ❸
      }
    ]'

```

- ❶ 使用在创建 `rawCNICConfig` 时提供的 `<name>`。
- ❷ 提供包括子网掩码的 IP 地址。
- ❸ 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
```

16.6. 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。

16.6.1. 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。

16.7. 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。

16.7.1. 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"]}}} }
```

16.8. 删除额外网络

作为集群管理员，您可以删除额外网络附加。

16.8.1. 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。

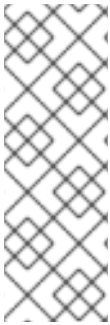
- 保存您的更改，再退出文本编辑器以提交更改。
- 可选：通过运行以下命令确认删除了额外网络 CR：

```
$ oc get network-attachment-definition --all-namespaces
```

16.9. 为 VRF 分配从属网络

16.9.1. 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 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。



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，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。

16.9.1.1. 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。

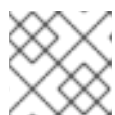
流程

- 为额外网络附加创建 **Network** 自定义资源 (CR)，并为额外网络插入 **rawCNICConfig** 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", 2
          "master": "eth1",
          "ipam": {
            "type": "static",
            "addresses": [
              {
                "address": "191.168.1.23/24"
              }
            ]
          }
        },
        {
          "type": "vrf",
          "vrfname": "example-vrf-name", 3
          "table": 1001 4
        }
      ]
    }'
```

- 1 **plugins** 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 可选的。**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. 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。

验证额外的 VRF 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网络附加，请执行以下操作：

1. 创建使用 VRF CNI 的网络。
2. 将网络分配给 pod。
3. 验证 Pod 网络附加是否已连接到 VRF 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
```

第 17 章 硬件网络

17.1. 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"
```

17.1.1. 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。

17.1.1.1. 支持的平台

在以下平台上支持 SR-IOV Network Operator：

- 裸机
- Red Hat OpenStack Platform(RHOSP)

17.1.1.2. 支持的设备

OpenShift Container Platform 支持以下网络接口控制器：

表 17.1. 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
Intel	X710	8086	1572
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
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

制造商	model	供应商 ID	设备 ID
Mellanox	MT28880 系列 [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 系列 [ConnectX-6]	15b3	101b
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d



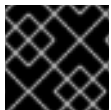
注意

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](#)。

17.1.1.3. 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。

17.1.1.3.1. SriovNetworkNodeState 对象示例

以下 YAML 是 SR-IOV Network Operator 创建的 `SriovNetworkNodeState` 对象示例：

一个 `SriovNetworkNodeState` 对象

```
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。

17.1.1.4. 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

```

17.1.1.5. 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 pod 规格中的环境变量，容器镜像可以运行以下 DPDK 示例应用程序之一：**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。

17.1.1.6. 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，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。

17.1.2. 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](#)

17.2. 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。

17.2.1. 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 SR-IOV Network Operator。

17.2.1.1. 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** 权限的帐户。

流程

1. 运行以下命令来创建 **openshift-sriov-network-operator** 命名空间：

```
$ 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. 订阅 SR-IOV Network Operator。

- a. 运行以下命令以获取 OpenShift Container Platform 的主版本和次版本。下一步中需要 **channel** 值。

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
  grep -o '[0-9]*[.][0-9]*' | head -1)
```

- b. 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: "${OC_VERSION}"
  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.4.10.0-202110121402  Succeeded
```

17.2.1.2. 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** 权限的帐户。

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**。

17.2.2. 后续步骤

- 可选：[配置 SR-IOV Network Operator](#)

17.3. 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。

17.3.1. 配置 SR-IOV Network Operator



重要

通常不需要修改 SR-IOV Network Operator 配置。我们推荐在大多数用例中使用默认配置。只有 Operator 的默认行为与您的用例不兼容时，才需要按照以下步骤修改相关配置。

SR-IOV Network Operator 添加了 **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition 资源。Operator 会在 **openshift-sriov-network-operator** 命名空间中自动创建一个名为 **default** 的 SriovOperatorConfig 自定义资源 (CR)。



注意

default CR 包含集群的 SR-IOV Network Operator 配置。要更改 Operator 配置，您必须修改此 CR。

17.3.1.1. 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 17.2. SR-IOV Network Operator 配置自定义资源

字段	类型	描述
metadata.name	字符串	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.name space	字符串	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	字符串	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 。

17.3.1.2. 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 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

17.3.1.3. 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

17.3.1.4. 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。

17.3.1.5. 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>
```

17.3.1.6. 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** 来启用它：

```
$ 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>

```

17.3.1.7. 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上重新创建。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>

```

17.3.1.8. 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**，请输入以下命令：

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{"spec": {"disableDrain": true }}'
```

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
```

17.3.2. 后续步骤

- [配置 SR-IOV 网络设备](#)

17.4. 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。

17.4.1. 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
  nicSelector: 9
    vendor: "<vendor_code>" 10
    deviceID: "<device_id>" 11
    pfNames: ["<pf_name>", ...] 12
    rootDevices: ["<pci_bus_id>", ...] 13
    netFilter: "<filter_string>" 14
  deviceType: <device_type> 15
  isRdma: false 16
  linkType: <link_type> 17
  eSwitchMode: "switchdev" 18

```

- 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会暂停，直到 MCP 返回健康状态。

为了避免不健康的 MCP 中的节点阻止将节点网络配置策略应用到其他节点，包括其他 MCP 中的节点，您必须为每个 MCP 创建单独的节点网络配置策略。

- 5 可选：priority（优先级）是 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 高于 99 的优先级。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。



重要

如果要在默认网络接口上创建虚拟功能，请确保将 MTU 设置为与集群 MTU 匹配的值。

- 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 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。如果您同时指定了 **pfNames** 和 **rootDevices**，请确保它们引用同一设备。如果您为 **netFilter** 指定一个值，则不需要指定任何其他参数，因为网络 ID 是唯一的。

- 10 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**。
- 11 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。
- 12 可选：该设备的一个或多个物理功能 (PF) 名称的数组。
- 13 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**。
- 14 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-xxxxxxxxxxxxxx**。将 **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx** 替换为 **/var/config/openstack/latest/network_data.json** 元数据文件中的值。
- 15 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**。

- 16 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。

- 17 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** 设置为 **ib** 时，**deviceType** 不应该设置为 **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。

- 18 可选：要启用硬件卸载，必须将 'eSwitchMode' 字段设置为 **"switchdev"**。

17.4.1.1. 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** 的有效值可从 **SriovNetworkNodeState** 对象获得。

17.4.1.2. 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，您可能希望某些 VF 使用默认驱动程序加载，剩余的 VF 负载使用 **vfio-pci** 驱动程序。在这种情况下，您的 **SriovNetworkNodePolicy** 自定义资源 (CR) 中的 **pfNames** 选择器可以用来使用以下格式为池指定 VF 范围：**<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，**numVfs** 设置为 **8**，则 **<first_vf>** 的值不能小于 **0**，**<last_vf>** 不得大于 **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
```

17.4.2. 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

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。

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。将 **<name>** 替换为此配置的名称。

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>** 指定此配置的名称。

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](#)

17.4.3. 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。

17.4.4. 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，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。

17.4.4.1. 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: intelnic
  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

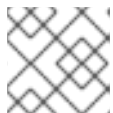
- 运行以下命令，确认 SR-IOV Network Operator 创建了 **NetworkAttachmentDefinition** CR。

```
$ 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：

- 创建使用 VRF CNI 的 SR-IOV 网络。
- 将网络分配给 pod。
- 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
```

- 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
...
```

17.4.5. 后续步骤

- [配置 SR-IOV 网络附加](#)

17.5. 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。

17.5.1. 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。整数值必须从 0 到 4095。默认值为 0。

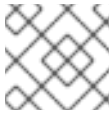
- 6 可选：VF 的 spoof 检查模式。允许的值是字符串 **"on"** 和 **"off"**。



重要

指定的值必须由引号包括，否则 SR-IOV Network Operator 将拒绝对象。

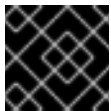
- 7 为 IPAM CNI 插件指定一个配置对象做为一个 YAML 块 scalar。该插件管理附加定义的 IP 地址分配。
- 8 可选：虚拟功能（VF）的连接状态。允许的值是 **enable**、**disable** 和 **auto**。
- 9 可选：VF 的最大传输率（以 Mbps 为单位）。
- 10 可选：VF 的最低传输率（以 Mbps 为单位）。这个值必须小于或等于最大传输率。



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。

17.5.1.1. 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。

17.5.1.1.1. 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 17.3. ipam 静态配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。值必须是 static 。

字段	类型	描述
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 17.4. ipam.addresses[] array

字段	类型	描述
address	字符串	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	字符串	出口网络流量要路由到的默认网关。

表 17.5. ipam.routes[] array

字段	类型	描述
dst	字符串	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	字符串	网络流量路由的网关。

表 17.6. 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"
    }
  ]
}
}

```

17.5.1.1.2. 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 17.7. ipam DHCP 配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。需要值 dhcp 。

动态 IP 地址(DHCP)分配配置示例

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

17.5.1.1.3. 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。

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置：

表 17.8. ipam whereabouts 配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。需要 abouts 的值。
range	字符串	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 标记中零个或多个 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。

使用 Whereabouts 的动态 IP 地址分配配置示例

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

17.5.1.1.4. 创建 Whereabouts 协调器守护进程集

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** 自定义资源进行动态 IP 地址分配。

通过 Cluster Network Operator 配置额外网络时，Whereabouts reconciler 守护进程集会被自动创建。从 YAML 清单配置额外网络时，它不会自动创建。

要触发 Whereabouts 协调器 daemonset 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 **whereabouts-shim** 网络附加。

使用以下步骤部署 Whereabouts 协调器 daemonset。

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. 修改 CR 中的 **additionalNetworks** 参数，以添加 **abouts-shim** 网络附加定义。例如：

```
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-p4sxx 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
```

17.5.2. 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>
```

17.5.3. 后续步骤

- [将 pod 添加到额外网络](#)

17.5.4. 其他资源

- [配置 SR-IOV 网络设备](#)

17.6. 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。

17.6.1. 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，或者 **"{ "infinibandGUID": true }"** 启用 IB Global Unique Identifier（GUID）支持。

17.6.1.1. 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。

17.6.1.1.1. 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 17.9. ipam 静态配置对象

字段	类型	描述
type	字符串	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 17.10. ipam.addresses[] array

字段	类型	描述
address	字符串	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	字符串	出口网络流量要路由到的默认网关。

表 17.11. ipam.routes[] array

字段	类型	描述
dst	字符串	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	字符串	网络流量路由的网关。

表 17.12. 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"
      }
    ]
  }
}
```

17.6.1.1.2. 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 17.13. ipam DHCP 配置对象

字段	类型	描述
type	字符串	IPAM 地址类型。需要值 dhcp 。

动态 IP 地址(DHCP)分配配置示例

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

17.6.1.1.3. 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。

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置：

表 17.14. ipam whereabouts 配置对象

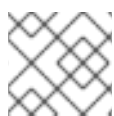
字段	类型	描述
type	字符串	IPAM 地址类型。需要 abouts 的值。
range	字符串	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	数组	可选：CIDR 标记中零个或多个 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。

使用 Whereabouts 的动态 IP 地址分配配置示例

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

17.6.1.1.4. 创建 Whereabouts 协调器守护进程集

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** 自定义资源进行动态 IP 地址分配。

通过 Cluster Network Operator 配置额外网络时，Whereabouts reconciler 守护进程集会被自动创建。从 YAML 清单配置额外网络时，它不会自动创建。

要触发 Whereabouts 协调器 daemonset 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 **whereabouts-shim** 网络附加。

使用以下步骤部署 Whereabouts 协调器 daemonset。

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. 修改 CR 中的 **additionalNetworks** 参数，以添加 **abouts-shim** 网络附加定义。例如 :

```
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-p4sxx 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
```

17.6.2. 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>
```

17.6.3. 后续步骤

- [将 pod 添加到额外网络](#)

17.6.4. 其他资源

- [配置 SR-IOV 网络设备](#)

17.7. 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。

17.7.1. 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/networks`，它接受描述运行时配置的 JSON 对象。

17.7.1.1. 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"]
```


17.7.1.2. 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>", ❶
    "infiniband-guid": "<guid>", ❷
    "ips": ["<cidr_range>"] ❸
  }
]
```

- ❶ SR-IOV 网络附加定义 CR 的名称。
- ❷ 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** }。
- ❸ 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

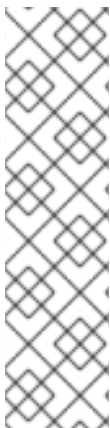
```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

17.7.2. 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/networks-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/networks-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。

17.7.3. 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**。

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> ❶
spec:
  containers:
    - name: sample-container
      image: <image> ❷
      command: ["sleep", "infinity"]
      resources:
        limits:
          memory: "1Gi" ❸
          cpu: "2" ❹
        requests:
          memory: "1Gi"
          cpu: "2"
```

- ❶ 将 `<name>` 替换为 SR-IOV 网络附加定义 CR 的名称。
- ❷ 将 `<image>` 替换为 **sample-pod** 镜像的名称。
- ❸ 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。
- ❹ 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> ❶
```

- ❶ 将 `<filename>` 替换为您在上一步中创建的文件名称。

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
```

17.7.4. 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](#)

17.8. 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。

17.8.1. 高性能多播

OpenShift SDN 默认 Container Network Interface (CNI) 网络供应商支持默认网络上的 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。

17.8.2. 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"] ❶
command: ["sleep", "infinity"]
```

- ❶ 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。

17.9. 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](#)。

17.9.1. 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 ❶
```

- ❶ 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。

- 运行以下命令来创建 `SriovNetworkNodePolicy` 对象：

```
$ oc create -f intel-dpdk-node-policy.yaml
```

- 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: intelnic
```

- 为 ipam CNI 插件指定一个配置对象作为一个 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。

- 运行以下命令来创建 `SriovNetwork` 对象：

```
$ oc create -f intel-dpdk-network.yaml
```

- 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> ❷
    securityContext:
      runAsUser: 0
    capabilities:
      add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] ❸
    volumeMounts:
    - mountPath: /dev/hugepages ❹
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" ❺
        memory: "1Gi"
        cpu: "4" ❻
        hugepages-1Gi: "4Gi" ❼
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
    volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- ❶ 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`。
- ❷ 指定包含应用程序和应用程序使用的 DPDK 库的 DPDK 镜像。
- ❸ 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。
- ❹ 在 `/dev/hugepages` 下将巨页卷挂载到 DPDK pod。巨页卷由 `emptyDir` 卷类型支持，媒介是 `Hugepages`。
- ❺ 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。
- ❻ 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。
- ❼ 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` 将导致系统引导过程中分配 `16*1Gi` 巨页。

6. 运行以下命令来创建 DPDK pod:

```
$ oc create -f intel-dpdk-pod.yaml
```

17.9.2. 在带有 Mellanox NIC 的 DPDK 模式中使用虚拟功能

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 **mlx-dpdk-node-policy.yaml** 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  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。Mellanox 卡允许的值是 **1015**，**1017**。
- ② 指定到 **netdevice** 的虚拟功能 (VF) 的驱动器类型。Mellanox SR-IOV 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出现。
- ③ 启用 RDMA 模式。Mellanox 卡需要在 DPDK 模式下工作。



注意

如需了解 **inSriovNetworkNodePolicy** 中的每个选项的信息，请参阅 **配置 SR-IOV 网络设备** 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-dpdk-node-policy.yaml
```

3. 创建以下 **SriovNetwork** 对象，然后在 **mlx-dpdk-network.yaml** 文件中保存 YAML。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ❶
# ...
  vlan: <vlan>
  resourceName: mlxnic
```

- ❶ 为 ipam CNI 插件指定一个配置对象作为一个 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-dpdk-network.yaml
```

5. 创建以下 **Pod** spec，然后在 **mlx-dpdk-pod.yaml** 文件中保存 YAML。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> ❷
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ❸
    volumeMounts:
  - mountPath: /dev/hugepages ❹
    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

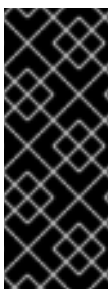
```

- 1 指定 **target_namespace**，它与 **SriovNetwork** 对象 **mlx-dpdk-network** 创建于的命令空间相同。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 **/dev/hugepages** 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。

6. 运行以下命令来创建 DPDK pod:

```
$ oc create -f mlx-dpdk-pod.yaml
```

17.9.3. 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: mlxnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceId: "1015" ❶
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice ❷
  isRdma: true ❸
```

- ❶ 指定 SR-IOV 网络设备的设备十六进制代码。Mellanox 卡允许的值是 **1015**，**1017**。
- ❷ 指定到 **netdevice** 的虚拟功能 (VF) 的驱动程序类型。
- ❸ 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: |- ❶
  # ...
  vlan: <vlan>
  resourceName: mlxnic
```

- ❶ 为 ipam CNI 插件指定一个配置对象作为一个 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> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> ❷
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ❸
    volumeMounts:
  - mountPath: /dev/hugepages ❹
    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 `/dev/hugepages` 下将巨页卷挂载到 RDMA pod。巨页卷由 `emptyDir` 卷类型支持，媒介是 `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
```

17.9.4. 其他资源

- [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。

17.10. 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](#)。

17.10.1. 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`。

17.10.1.1. 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。

17.10.1.2. 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
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 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 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 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

```

17.11. 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。

17.11.1. 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。

17.11.2. 支持的设备

在以下网络接口控制器上支持硬件卸载：

表 17.15. 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

17.11.3. 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 CNI](#)。
- 在 [OVN-Kubernetes CNI 配置中](#)，`gatewayConfig.routingViaHost` 字段被设置为 `false`。

17.11.4. 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 ❶
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,mcp-
offloading]} ❷
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-offloading: "" ❸

```

❶ ❷ 用于硬件卸载的机器配置池的名称。

❸ 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   v1.23.3+d99c04f
master-1  Ready  master         2d   v1.23.3+d99c04f
master-2  Ready  master         2d   v1.23.3+d99c04f
worker-0  Ready  worker         2d   v1.23.3+d99c04f
worker-1  Ready  worker         2d   v1.23.3+d99c04f
worker-2  Ready  mcp-offloading,worker  47h  v1.23.3+d99c04f
worker-3  Ready  mcp-offloading,worker  47h  v1.23.3+d99c04f

```

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 ❶

```

❶ 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。

17.11.5. 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 <.>
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice <.>
  eSwitchMode: "switchdev" <.>
  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: mlxnic
```

<.> 自定义资源对象的名称。<.> 必需。**vfio-pci** 不支持硬件卸载。<.> 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。

17.11.6. 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 <.>
  namespace: net-attach-def <.>
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnic <.>
spec:
  config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":
  {}, "dns":{}}'
```

<.> 网络附加定义的名称。<.> 网络附加定义的命名空间。<.> 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
```

17.11.7. 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 <.>
```

<.> 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。

17.12. 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。

17.12.1. 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

- 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
```

- 按照 "Deleting Operators from a cluster" 部分的说明从集群中移除 SR-IOV Network Operator。
- 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 18 章 OPENSIFT SDN 默认 CNI 网络供应商

18.1. 关于 OPENSIFT SDN 默认 CNI 网络供应商

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。此 pod 网络是由 OpenShift SDN 建立和维护的，它使用 Open vSwitch (OVS) 配置覆盖网络。

18.1.1. 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 4.10 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。

18.1.2. 支持的默认 CNI 网络供应商功能列表

OpenShift Container Platform 为默认的 Container Network Interface (CNI) 网络供应商提供两个支持的选择：OpenShift SDN 和 OVN-Kubernetes。下表总结了这两个网络供应商当前支持的功能：

表 18.1. 默认 CNI 网络供应商功能比较

功能	OpenShift SDN	OVN-Kubernetes
出口 IP	支持	支持
出口防火墙 ^[1]	支持	支持
出口路由器	支持	支持 ^[2]
IPsec 加密	不支持	支持
IPv6	不支持	支持 ^{[3][4]}
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. IPv6 只在裸机集群中被支持。
4. IPv6 单堆栈不支持 [Kubernetes NMState](#)。

18.2. 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)集群网络供应商，为项目分配一个或多个出口 IP 地址。

18.2.1. 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`。

18.2.1.1. 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)	否
Amazon Web Services (AWS)	是
Google Cloud Platform (GCP)	是
Microsoft Azure	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)

18.2.1.2. 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：

$$\text{IP capacity} = \text{public cloud default capacity} - \text{sum}(\text{current IP assignments})$$

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。

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。

18.2.1.2.1. 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](#)

18.2.1.2.2. 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 10。
- 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](#)。

18.2.1.2.3. 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](#)。

18.2.1.3. 限制

将出口 IP 地址与 OpenShift SDN 集群网络供应商搭配使用时会有以下限制：

- 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 集群网络供应商出口 IP 地址可以在多个命名空间中分散 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](#)。

18.2.1.4. 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。

18.2.1.4.1. 使用自动分配的出口 IP 地址时的注意事项

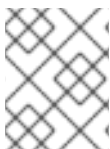
当对出口 IP 地址使用自动分配方法时，请注意以下事项：

- 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。

18.2.1.4.2. 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。

18.2.2. 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。

- 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。

18.2.3. 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。

18.2.4. 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。

18.3. 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。

18.3.1. 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。为确保 pod 可以访问 OpenShift Container Platform API 服务器，您必须包含内置加入 Open Virtual Network(OVN)的网络 **100.64.0.0/16** 网络，以便在将节点端口与 EgressFirewall 搭配使用时允许访问。您还必须包含 API 服务器在出口防火墙规则中侦听的 IP 地址范围，如下例所示：

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
  namespace: <namespace> ❶
spec:
  egress:
    - to:
      cidrSelector: <api_server_address_range> ❷
      type: Allow
    # ...
    - to:
      cidrSelector: 0.0.0.0/0 ❸
      type: Deny
```

- ❶ 出口防火墙的命名空间。
- ❷ 包含 OpenShift Container Platform API 服务器的 IP 地址范围。
- ❸ 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。

18.3.1.1. 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。
- 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 默认 Container Network Interface (CNI) 网络供应商时，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，并可能导致所有外部网络流量被丢弃。

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

18.3.1.2. 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。

18.3.1.3. 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

出口防火墙始终允许 pod 访问 pod 所在的用于 DNS 解析的节点的外部接口。

如果您在出口防火墙策略中使用域名，且您的 DNS 解析不是由本地节点上的 DNS 服务器处理，那么您必须添加出口防火墙规则，允许访问您的 DNS 服务器的 IP 地址。如果您在 pod 中使用域名。

18.3.2. 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> ❶
spec:
  egress: ❷
  ...
```

- ❶ 出口防火墙的名称。
- ❷ 以下部分所述，一个或多个出口网络策略规则的集合。

18.3.2.1. EgressNetworkPolicy 规则

以下 YAML 描述了一个出口防火墙规则对象。**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> ❶
  to: ❷
    cidrSelector: <cidr> ❸
    dnsName: <dns_name> ❹
```

- ❶ 规则类型。该值必须是 **Allow** 或 **Deny**。
- ❷ 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。
- ❸ CIDR 格式的 IP 地址范围。
- ❹ 一个域名。

18.3.2.2. 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: ❶
  - 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。

18.3.3. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 安装 OpenShift CLI (**oc**)。
- 您需要使用集群管理员身份登录到集群。

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。

18.4. 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。

18.4.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 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
```

18.5. 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。

18.5.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 安装 OpenShift CLI (**oc**)。
- 您需要使用集群管理员身份登录到集群。

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
```

18.6. 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。

18.6.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 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
```

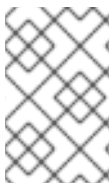
2. 输入以下命令删除 EgressNetworkPolicy 对象。将 **<project>** 替换为项目名称，**<name>** 替换为对象名称。

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

18.7. 使用出口路由器 POD 的注意事项

18.7.1. 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。

18.7.1.1. 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。

18.7.1.2. 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。

18.7.1.3. 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>
```

Red Hat Virtualization (RHV)

如果使用 [RHV](#)，必须为虚拟网络接口控制器 (vNIC) 选择 **No Network Filter**。

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](#)

18.7.1.4. 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，请使用 **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。

18.7.2. 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

18.8. 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。

18.8.1. 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，到 pod 的连接会被重定向到 **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

```

18.8.2. 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>** - 如上所示，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
```

18.8.3. 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。

18.8.4. 其他资源

- [使用 ConfigMap 配置出口路由器目的地映射](#)

18.9. 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。

18.9.1. 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" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE ❷
        value: <egress-router>
      - name: EGRESS_GATEWAY ❸
        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 ❹
        value: |-
          ...
          ...
```

- ❶ 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**。
- ❷ 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。
- ❸ 值与节点使用的默认网关相同。
- ❹ 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。

18.9.2. 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
*
```

18.9.3. 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**。

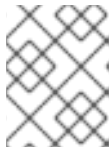
3. 要将客户端 pod（不是出口代理 Pod）配置为使用 HTTP 代理，设置 **http_proxy** 或 **https_proxy** 变量：

```

apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
      - name: http_proxy
        value: http://egress-1:8080/ ❶
      - name: https_proxy
        value: http://egress-1:8080/
      ...

```

❶ 上一步中创建的服务。



注意

不是所有的设置都需要使用 **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。

18.9.4. 其他资源

- [使用 ConfigMap 配置出口路由器目的地映射](#)

18.10. 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。

18.10.1. 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" ❶
spec:
  initContainers:
    - name: egress-router
      image: registry.redhat.io/openshift4/ose-egress-router
      securityContext:
        privileged: true
  env:
    - name: EGRESS_SOURCE ❷
      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**。

18.10.2. 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
```

18.10.3. 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，其中包含以下 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。

18.10.4. 其他资源

- [使用 ConfigMap 配置出口路由器目的地映射](#)

18.11. 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。

18.11.1. 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
...
```

18.11.2. 其他资源

- [重定向模式](#)
- [HTTP 代理模式](#)
- [DNS 代理模式](#)

18.12. 为项目启用多播

18.12.1. 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，OpenShift Container Platform pod 之间多播流量被禁用。如果使用 OpenShift SDN 默认 Container Network Interface (CNI) 网络供应商，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。

18.12.2. 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/ubi8
    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/ubi8
    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
```

18.13. 为项目禁用多播

18.13.1. 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**。

18.14. 使用 OPENSIFT SDN 配置网络隔离

将集群配置为使用 OpenShift SDN CNI 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。

18.14.1. 先决条件

- 您必须将集群配置为以多租户隔离模式使用 OpenShift SDN Container Network Interface (CNI) 插件。

18.14.2. 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。

18.14.3. 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。

18.14.4. 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。

18.15. 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。

18.15.1. 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。

18.15.2. 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 18.2. 参数

参数	描述	值	默认值
<code>iptablesSyncPeriod</code>	<code>iptables</code> 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
<code>proxyArguments.iptables-min-sync-period</code>	刷新 <code>iptables</code> 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 <code>iptables</code> 规则的更改时立即进行刷新。	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

18.15.3. 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` 角色登录到正在运行的集群。

续前

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 19 章 OVN-KUBERNETES 默认 CNI 网络供应商

19.1. 关于 OVN-KUBERNETES 默认 CONTAINER NETWORK INTERFACE (CNI) 网络供应商

OpenShift Container Platform 集群在 pod 和服务网络中使用虚拟网络。OVN-Kubernetes Container Network Interface (CNI) 插件是默认集群网络的一个网络供应商。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 网络供应商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。

19.1.1. OVN-Kubernetes 特性

OVN-Kubernetes Container Network Interface (CNI) 集群网络供应商实现以下功能：

- 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。

19.1.2. 支持的默认 CNI 网络供应商功能列表

OpenShift Container Platform 为默认的 Container Network Interface (CNI) 网络供应商提供两个支持的选择：OpenShift SDN 和 OVN-Kubernetes。下表总结了这两个网络供应商当前支持的功能：

表 19.1. 默认 CNI 网络供应商功能比较

功能	OVN-Kubernetes	OpenShift SDN
出口 IP	支持	支持
出口防火墙 ^[1]	支持	支持
出口路由器	支持 ^[2]	支持
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. IPv6 只在裸机集群中被支持。
4. IPv6 单堆栈不支持 [Kubernetes NMState](#)。

19.1.3. OVN-Kubernetes 限制

OVN-Kubernetes Container Network Interface(CNI)集群网络供应商有以下限制：

- OVN-Kubernetes 不支持将 Kubernetes 服务的内部流量策略设置为 **local**。当您添加类型为 **ClusterIP**、**LoadBalancer**、**NodePort** 或使用外部 IP 的服务时，这个限制可能会影响到应用程序的网络通信。
- **sessionAffinityConfig.clientIP.timeoutSeconds** 服务在 OpenShift OVN 环境中无效，但在 OpenShift SDN 环境中不起作用。这种不兼容可能会导致用户难以从 OpenShift SDN 迁移到 OVN。
- 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。

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](#)
- [IPsec 加密配置](#)
- [Network \[operator.openshift.io/v1\]](#)

19.2. 从 OPENSIFT SDN 集群网络供应商迁移

作为集群管理员，您可以从 OpenShift SDN CNI 集群网络供应商迁移到 OVN-Kubernetes Container Network Interface (CNI) 集群网络供应商。

要了解更多有关 OVN-Kubernetes 的信息，请参阅[关于 OVN-Kubernetes 网络供应商](#)。

19.2.1. 迁移到 OVN-Kubernetes 网络供应商

迁移到 OVN-Kubernetes Container Network Interface (CNI) 集群网络供应商是一个手动过程，其中会包括一些停机时间使集群无法访问。虽然提供了一个回滚过程，但迁移通常被认为是一个单向过程。

在以下平台上支持迁移至 OVN-Kubernetes 集群网络供应商：

- 裸机硬件
- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- Red Hat OpenStack Platform(RHOSP)
- Red Hat Virtualization (RHV)
- VMware vSphere



重要

Red Hat OpenShift Dedicated, Azure Red Hat OpenShift(ARO), 和 Red Hat OpenShift Service on AWS (ROSA) 上的受管 OpenShift 云服务不支持迁移到 OVN-Kubernetes 网络插件。

19.2.1.1. 迁移到 OVN-Kubernetes 网络供应商时的注意事项

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 网络供应商实现了 OpenShift SDN 网络供应商中的许多功能，但配置并不相同。

- 如果您的集群使用以下 OpenShift SDN 功能，则必须在 OVN-Kubernetes 中手动配置相同的功能：
 - 命名空间隔离
 - 出口 IP 地址
 - 出口网络策略
 - 出口路由器 pod
 - 多播

- 如果您的集群使用 **100.64.0.0/16** IP 地址范围中的任何部分，则无法迁移到 OVN-Kubernetes，因为它在内部使用这个 IP 地址范围。

以下小节重点介绍了上述功能在 OVN-Kubernetes 和 OpenShift SDN 中的配置的不同。

命名空间隔离

OVN-Kubernetes 仅支持网络策略隔离模式。



重要

如果您的集群使用在多租户或子网隔离模式中配置的 OpenShift SDN，则无法迁移到 OVN-Kubernetes 网络供应商。

出口 IP 地址

下表中描述了在 OVN-Kubernetes 和 OpenShift SDN 配置出口 IP 地址的不同：

表 19.2. 出口 IP 地址配置的不同

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● 创建 EgressIPs 对象 ● 在一个 Node 对象上添加注解 	<ul style="list-style-type: none"> ● 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 19.3. 出口网络策略配置的不同

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● 在命名空间中创建 EgressFirewall 对象 	<ul style="list-style-type: none"> ● 在命名空间中创建一个 EgressNetworkPolicy 对象

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 19.4. 多播配置的不同

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> 在 Namespace 对象上添加注解 	<ul style="list-style-type: none"> 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 Kubernetes **NetworkPolicy** API。从 OpenShift SDN 进行迁移时，网络策略不需要更改。

19.2.1.2. 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 19.5. 从 OpenShift SDN 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。
更新 Network.config.openshift.io CR 的 networkType 字段。	CNO 执行以下操作： <ul style="list-style-type: none"> 销毁 OpenShift SDN control plane pod。 部署 OVN-Kubernetes control plane pod。 更新 Multus 对象以反映新的集群网络供应商。
重新引导集群中的每个节点。	集群 当节点重启时，集群会为 OVN-Kubernetes 集群网络上的 pod 分配 IP 地址。

如果需要回滚到 OpenShift SDN，下表描述了这个过程。

表 19.6. 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： <ul style="list-style-type: none"> ● 销毁 OVN-Kubernetes control plane pod。 ● 部署 OpenShift SDN control plane pod。 ● 更新 Multus 对象以反映新的集群网络供应商。
重新引导集群中的每个节点。	集群 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。

19.2.2. 迁移至 OVN-Kubernetes 默认 CNI 网络供应商

作为集群管理员，您可以将集群的默认 Container Network Interface (CNI) 网络供应商更改为 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下，使用 OpenShift SDN CNI 集群网络供应商配置的集群。
- 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。
- 在更新软件之后的所有云平台上，必须设置安全组规则，以允许 UDP 数据包在所有节点的端口 **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)
- Geneve (Generic Network Virtualization Encapsulation) 覆盖网络端口

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>
      }
    }
  }
}'
```

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**。

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
      }
    }
  }
}'
```

4. 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。

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
```

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。

6. 要启动迁移，请使用以下命令配置 OVN-Kubernetes 集群网络供应商：

- 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。

- 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-`<xxxxxx>`** 的形式，其中 `<xxxxxx>` 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
```

- 要完成迁移，请重新引导集群中的每个节点。例如，您可以使用类似以下示例的 bash 脚本。这个脚本假定您可以使用 **ssh** 连接到每个主机，并将 **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
```

如果无法使用 ssh 访问，您可能无法通过基础架构供应商的管理门户重新引导每个节点。

- 确认迁移成功完成：

- 要确认 CNI 集群网络供应商是 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}'
```

- 要确认集群节点处于 **Ready** 状态，请输入以下命令：

```
$ oc get nodes
```

- 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。

- 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。

10. 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
```

19.2.3. 其他资源

- [OVN-Kubernetes 网络供应商的配置参数](#)
- [备份 etcd](#)
- [关于网络策略](#)
- 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](#)
- [Network \[operator.openshift.io/v1\]](#)

19.3. 回滚到 OPENSIFT SDN 网络供应商

作为集群管理员，如果迁移到 OVN-Kubernetes-Kubernetes 失败，您可以回滚到 OVN-Kubernetes CNI 集群网络供应商的 OpenShift SDN Container Network Interface (CNI) 集群网络供应商。

19.3.1. 将默认 CNI 网络供应商回滚到 OpenShift SDN

作为集群管理员，您可以将集群回滚到 OpenShift SDN Container Network Interface (CNI) 集群网络供应商。在回滚过程中，您必须重新引导集群中的每个节点。



重要

只有迁移到 OVN-Kubernetes 失败时才会回滚到 OpenShift SDN。

先决条件

- 安装 OpenShift CLI (**oc**)。
- 使用具有 **cluster-admin** 角色的用户访问集群。
- 在使用 OVN-Kubernetes CNI 集群网络供应商配置的基础架构上安装集群。

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. 要开始迁移，请输入以下命令将集群网络供应商重新设置为 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" } }'
```

3. 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
      }
    }
  }
}'
```

4. 等待 Multus 守护进程集的 rollout 完成。

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus pod 的名称格式为 **multus-`<xxxxxx>`**，其中 **`<xxxxxxx>`** 是字母的随机序列。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
```

5. 要完成回滚，请重新引导集群中的每个节点。例如，您可以使用类似如下的 bash 脚本。这个脚本假定您可以使用 **ssh** 连接到每个主机，并将 **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
```

如果无法使用 ssh 访问，您可能无法通过基础架构供应商的管理门户重新引导每个节点。

6. 重新引导集群中的节点后，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。

7. 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。

8. 确认迁移成功完成：

- a. 要确认默认 CNI 网络供应商是 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}'
```

- 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。

9. 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：

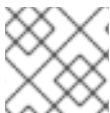
```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{"spec": {"defaultNetwork": {"ovnKubernetesConfig": null }}}'
```

- 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
```

19.4. 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。转换为双栈后，所有新创建的 pod 都启用了双栈。



注意

在裸机、IBM Power 基础架构和单一节点 OpenShift 集群上置备的集群上支持双栈网络。



注意

在使用双栈网络时，您无法使用 IPv4 映射 IPv6 地址，如 `::FFFF:198.51.100.1`，其中需要 IPv6。

19.4.1. 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。



注意

转换为双栈网络后，只有新创建的 pod 会被分配 IPv6 地址。必须重新创建在转换前创建的所有 pod，才能接收 IPv6 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 集群网络供应商。
- 集群节点具有 IPv6 地址。

流程

1. 要为集群和服务网络指定 IPv6 地址块，请创建一个包含以下 YAML 的文件：

```
- 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 CIDR 前缀必须足够大，以容纳指定的主机前缀。
- 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** 到 **128** 位进行分配。

2. 要修补集群网络配置，请输入以下命令：

```
$ oc patch network.config.openshift.io cluster \
  --type='json' --patch-file <file>.yaml
```

其中：

file

指定您在上一步中创建的文件名称。

输出示例

```
network.config.openshift.io/cluster patched
```

验证

完成以下步骤以验证，集群网络是否可以识别您在上一步中指定的 IPv6 地址块。

1. 显示网络配置：

```
$ oc describe network
```

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
```

19.5. IPSEC 加密配置

启用 IPsec，则 OVN-Kubernetes Container Network Interface (CNI) 集群网络中的所有节点之间的网络流量都可以通过加密的隧道进行。

默认禁用 IPsec。

**注意**

IPsec 加密只能在集群安装过程中启用，且在启用后无法禁用。有关安装文档，请参阅[选择集群安装方法并为用户准备它](#)。

19.5.1. 使用 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：

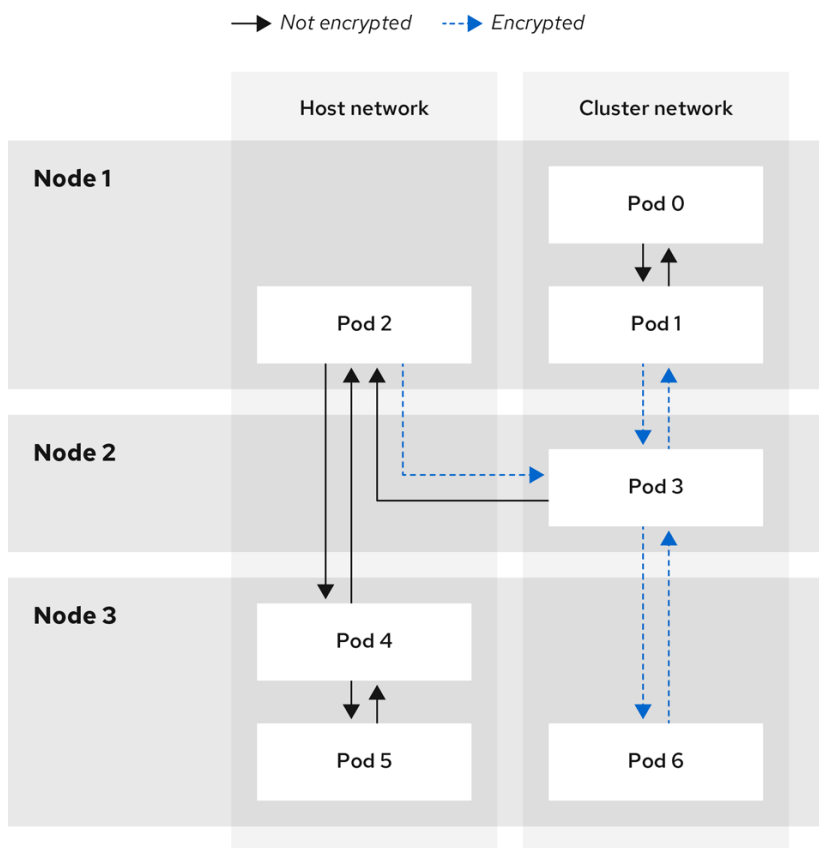
- 集群网络的不同节点上的 pod 间的流量
- 从主机网络上的 pod 流量到集群网络上的 pod

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：



138_OpenShift_0421

19.5.1.1. 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 19.7. 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)

19.5.2. 加密协议和 IPsec 模式

使用的加密密码是 **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*。

19.5.3. 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。

19.6. 为项目配置出口防火墙

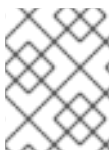
作为集群管理员，您可以为项目创建一个出口防火墙，用于限制离开 OpenShift Container Platform 集群的出口流量。

19.6.1. 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。为确保 pod 可以访问 OpenShift Container Platform API 服务器，您必须包含内置加入 Open Virtual Network(OVN)的网络 **100.64.0.0/16** 网络，以便在将节点端口与 EgressFirewall 搭配使用时允许访问。您还必须包含 API 服务器在出口防火墙规则中侦听的 IP 地址范围，如下例所示：

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

- ❶ 出口防火墙的命名空间。
- ❷ 包含 OpenShift Container Platform API 服务器的 IP 地址范围。
- ❸ 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。

19.6.1.1. 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，并可能导致所有外部网络流量被丢弃。

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

19.6.1.2. 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。

19.6.1.3. 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

出口防火墙始终允许 pod 访问 pod 所在的用于 DNS 解析的节点的外部接口。

如果您在出口防火墙策略中使用域名，且您的 DNS 解析不是由本地节点上的 DNS 服务器处理，那么您必须添加出口防火墙规则，允许访问您的 DNS 服务器的 IP 地址。如果您在 pod 中使用域名。

19.6.2. 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。

19.6.2.1. EgressFirewall 规则

以下 YAML 描述了一个出口防火墙规则对象。**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
  ports: ⑤
  ...
```

- 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 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**。

19.6.2.2. 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
```

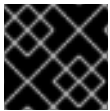
1 出口防火墙策略规则对象的集合。

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
```

19.6.3. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 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。

19.7. 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。

19.7.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 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
```

19.8. 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。

19.8.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。

- 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
```

19.9. 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。

19.9.1. 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 默认 Container Network Interface (CNI) 网络供应商插件的集群。
- 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>
```

19.10. 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)集群网络供应商，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。

19.10.1. 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`。

19.10.1.1. 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)	否
Amazon Web Services (AWS)	是
Google Cloud Platform (GCP)	是
Microsoft Azure	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)

19.10.1.2. 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：

$$\text{IP capacity} = \text{public cloud default capacity} - \text{sum}(\text{current IP assignments})$$

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。

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。

19.10.1.2.1. 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](#)

19.10.1.2.2. 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 10。

- 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](#)。

19.10.1.2.3. 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](#)。

19.10.1.3. 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。

19.10.1.4. 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：

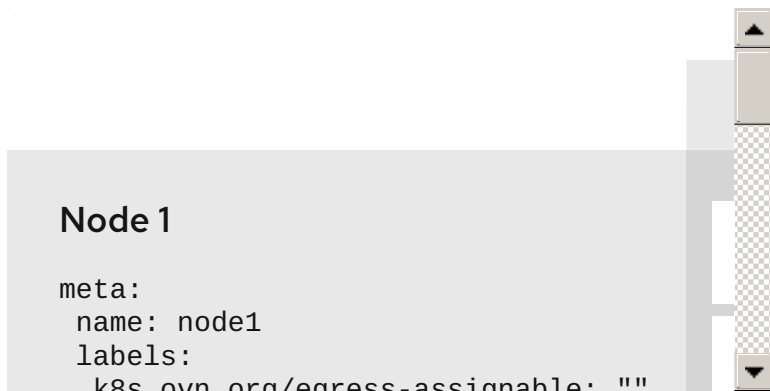
- 每次不会将出口 IP 地址分配给多个节点。
- 出口 IP 地址可在可以托管出口 IP 地址的可用节点之间平衡。
- 如果 **EgressIP** 对象中的 `spec.EgressIPs` 数组指定了多个 IP 地址，则适用以下条件：
 - 任何节点都不会托管超过一个指定的 IP 地址。
 - 流量在给定命名空间的指定 IP 地址之间大致相等。
- 如果节点不可用，则会自动重新分配给它的所有出口 IP 地址，但符合前面描述的条件。

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。

19.10.1.5. 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：

Namespace 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，它选择将 **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儿分配了出口 IP 地址。

19.10.2. 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> ❶
spec:
  egressIPs: ❷
  - <ip_address>
  namespaceSelector: ❸
  ...
  podSelector: ❹
  ...

```

- ❶ **EgressIPs** 对象的名称。
- ❷ 包括一个或多个 IP 地址的数组。
- ❸ 出口 IP 地址与其关联的一个或多个命名空间选择器将。
- ❹ 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: ❶
  matchLabels:
    <label_name>: <label_value>

```

- ❶ 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
```

19.10.3. 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>
```

19.10.4. 后续步骤

- [分配出口 IP](#)

19.10.5. 其他资源

- [labelSelector meta/v1](#)
- [LabelSelectorRequirement meta/v1](#)

19.11. 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。

19.11.1. 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 ❶
```

- ❶ 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 ❶
```

- ❶ 将 `<namespace>` 替换为需要出口 IP 地址的命名空间。

19.11.2. 其他资源

- [配置出口 IP 地址](#)

19.12. 使用出口路由器 POD 的注意事项

19.12.1. 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。

19.12.1.1. 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。

19.12.1.2. 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，并且具有该网络的 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。

19.12.1.3. 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>
```

Red Hat Virtualization (RHV)

如果使用 [RHV](#)，必须为虚拟网络接口控制器 (vNIC) 选择 **No Network Filter**。

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](#)

19.12.1.4. 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
```

19.12.2. 其他资源

- [在重定向模式中部署出口路由器](#)

19.13. 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。

19.13.1. 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> <.>
spec:
  addresses: [ <.>
    {
      ip: "<egress_router>", <.>
      gateway: "<egress_gateway>" <.>
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [ <.>
      {
        destinationIP: "<egress_destination>",
        port: <egress_router_port>,
        targetPort: <target_port>, <.>
        protocol: <network_protocol> <.>
      },
      ...
    ],
    fallbackIP: "<egress_destination>" <.>
  }

```

<.> 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。

<.> **address** 字段指定要在第二个网络接口上配置的 IP 地址。

<.> **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。

<.> **gateway** 字段指定网络网关的 IP 地址。

<.> 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。

<.> 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。

<.> **protocol** 字段支持 TCP、UDP 或 SCTP。

<.> 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
      }
    ]
  }
}
```

19.13.2. 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 <.>
```

<.> 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 {lindex: 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，并源自保留源 IP 地址 **192.168.12.99**。pod 将所有其他流量路由到网关，地址为 **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
```

19.14. 为项目启用多播

19.14.1. 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，OpenShift Container Platform pod 之间多播流量被禁用。如果使用 OVN-Kubernetes 默认 Container Network Interface (CNI) 网络供应商，则可以根据每个项目启用多播。

19.14.2. 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/ubi8
    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/ubi8
    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
```

19.15. 为项目禁用多播

19.15.1. 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
```

19.16. 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：

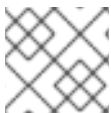
- NetFlow
- sFlow
- IPFIX

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。

19.16.1. 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 19.8. 网络流配置

字段	类型	描述
<code>metadata.name</code>	字符串	CNO 对象的名称。这个名称始终是 集群 。
<code>spec.exportNetworkFlows</code>	object	一个或多个 netFlow 、 sFlow 或 ipfix 。
<code>spec.exportNetworkFlows.netFlow.collectors</code>	数组	最多 10 个收集器的 IP 地址和网络端口对列表。
<code>spec.exportNetworkFlows.sFlow.collectors</code>	数组	最多 10 个收集器的 IP 地址和网络端口对列表。
<code>spec.exportNetworkFlows.ipfix.collectors</code>	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
```

19.16.2. 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-node $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"]-
    ...

```

19.16.3. 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

```

19.16.4. 其他资源

- [Network \[operator.openshift.io/v1\]](https://operator.openshift.io/v1)

19.17. 配置混合联网

作为集群管理员，您可以配置 OVN-Kubernetes Container Network Interface(CNI)集群网络供应商，允许 Linux 和 Windows 节点分别托管 Linux 和 Windows 工作负载。

19.17.1. 使用 OVN-Kubernetes 配置混合网络

您可以将集群配置为使用 OVN-Kubernetes 的混合网络。这允许支持不同节点网络配置的混合集群。例如：集群中运行 Linux 和 Windows 节点时需要这样做。



重要

您必须在安装集群过程中使用 OVN-Kubernetes 配置混合网络。您不能在安装过程中切换到混合网络。

先决条件

- 您在 `install-config.yaml` 文件中为 `networking.networkType` 参数定义了 `OVNKubernetes`。如需更多信息，请参阅有关在所选云供应商上配置 OpenShift Container Platform 网络自定义的安装文档。

流程

1. 进入包含安装程序的目录并创建清单：

```
$ ./openshift-install create manifests --dir <installation_directory>
```

其中：

<installation_directory>

指定包含集群的 `install-config.yaml` 文件的目录名称。

2. 在 `<installation_directory>/manifests/` 目录中为高级网络配置创建一个名为 `cluster-network-03-config.yml` 的 stub 清单文件：

```
$ cat <<EOF > <installation_directory>/manifests/cluster-network-03-config.yml
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
EOF
```

其中：

<installation_directory>

指定包含集群的 `manifests/` 目录的目录名称。

3. 在编辑器中打开 `cluster-network-03-config.yml` 文件，并使用混合网络配置 OVN-Kubernetes，如下例所示：

指定混合网络配置

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      hybridOverlayConfig:
```



```

hybridClusterNetwork: ❶
- cidr: 10.132.0.0/14
  hostPrefix: 23
hybridOverlayVXLANPort: 9898 ❷

```

- ❶ 指定用于额外覆盖网络上节点的 CIDR 配置。**hybridClusterNetwork** CIDR 无法与 **clusterNetwork** CIDR 重叠。
- ❷ 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](#)。



注意

Windows Server Long-Term Servicing Channel (LTSC) : Windows Server 2019 在带有自定义 **hybridOverlayVXLANPort** 值的集群中不被支持，因为这个 Windows server 版本不支持选择使用自定义的 VXLAN 端口。

4. 保存 **cluster-network-03-config.yml** 文件，再退出文本编辑器。
5. 可选：备份 **manifests/cluster-network-03-config.yml** 文件。创建集群时，安装程序会删除 **manifests/** 目录。

完成所有进一步的安装配置，然后创建集群。安装过程完成后会启用 Hybrid 网络。

19.17.2. 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 20 章 配置路由

20.1. 路由配置

20.1.1. 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。

先决条件

- 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
```

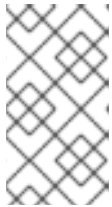
如果您检查生成的 **Route** 资源，它应该类似于如下：

创建的未安全路由的 YAML 定义：

```
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}
```

20.1.2. 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。

先决条件

- 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
```

20.1.3. 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。

20.1.3.1. 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。

先决条件

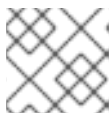
- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 `oc` CLI。

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，大约有 8 个半小时。



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**，它会对策略进行求值。
- 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。

20.1.3.2. 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**。

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 **oc** CLI。

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
```

- 要为命名空间中的每个路由禁用 HSTS，请输入 **lowinf** 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}}\n{{end}}\n{{end}}'
```

输出示例

```
Name: routename HSTS: max-age=0
```

20.1.3.3. 强制每个域 HTTP 严格传输安全性

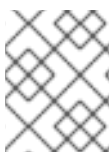
要为安全路由强制实施 HTTP Strict Transport Security (HSTS)，在 Ingress spec 中添加 **requiredHSTSPolicies** 记录来捕获 HSTS 策略的配置。

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。



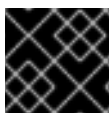
注意

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 **oc** CLI。

流程

1. 编辑 Ingress 配置文件：

```
$ 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: 7
- '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 7 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 可选的。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，否则如果您不会禁用 HSTS，请输入 1。它可以不知道，这意味着不强制实施较低限制。
- 5 可选的。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，才能获取标头。可使用以下之一设置 **preload**：
 - **RequirePreload** : **RequiredHSTSPolicy** 需要 **preload**。
 - **RequireNoPreload:preload** 被 **RequiredHSTSPolicy** 禁止。
 - **NoOpinion** : **preload** 与 **RequiredHSTSPolicy** 没有关系。
- 6 可选的。**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}}{\n}}{\end}}{\end}}
{{end}}'
```

输出示例

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

20.1.4. 吞吐量问题错误排解

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 日志未能揭示造成问题的原因，请使用以下方法分析性能问题：

- 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 在 **/tmp/dump.pcap** 中生成一个包含这两个 pod 间所有流量的文件。最好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数据流吞吐量和 UDP 吞吐量。先从 pod 运行该工具，再从节点运行，以此来查找瓶颈。
 - 如需有关安装和使用 iperf 的信息，请参阅此[红帽解决方案](#)。

20.1.5. 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 Controller 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。

20.1.5.1. 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。因此，如果服务器过载，它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。

- 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
```

20.1.6. 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。不过，这还取决于路由器的实现。

下表显示了路由及其可访问性示例：

表 20.1. 路由可用性

Route	当比较到	可访问
<i>www.example.com/test</i>	<i>www.example.com/test</i>	是
	<i>www.example.com</i>	否
<i>www.example.com/test</i> 和 <i>www.example.com</i>	<i>www.example.com/test</i>	是
	<i>www.example.com</i>	是
<i>www.example.com</i>	<i>www.example.com/text</i>	yes（由主机匹配，而不是路由）
	<i>www.example.com</i>	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" ❶
  to:
    kind: Service
    name: service-name
```

- 该路径是基于路径的路由的唯一添加属性。

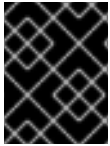


注意

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。

20.1.7. 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。

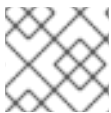
表 20.2. 路由注解

变量	描述	默认的环境变量
<code>haproxy.router.openshift.io/balancer</code>	设置负载均衡算法。可用选项是 random 、 source 、 roundrobin 和 leastconn 。默认值为 random 。	passthrough 路由 使用 ROUTER_TCP_BALANCE_SCHEME 。否则，使用 ROUTER_LOAD_BALANCE_algorithm 。
<code>haproxy.router.openshift.io/disable_cookies</code>	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。	
<code>router.openshift.io/cookie_name</code>	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。	
<code>haproxy.router.openshift.io/pod-concurrent-connections</code>	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。	
<code>haproxy.router.openshift.io/rate-limit-connections</code>	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：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	

变量	描述	默认的环境变量
haproxy.router.openshift.io/ate-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：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/ate-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：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/ate-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：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/ti meout	为路由设定服务器端超时。 (TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
haproxy.router.openshift.io/ti meout-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
<code>haproxy.router.openshift.io/ip_whitelist</code>	<p>为路由设置白名单。白名单是以空格分开的 IP 地址和 CIDR 范围列表，用来代表批准的源地址。来自白名单以外的 IP 地址的请求会被丢弃。</p> <p>白名单中允许的最大 IP 地址和 CIDR 范围数为 61。</p>	
<code>haproxy.router.openshift.io/https_header</code>	为 edge terminated 或 re-encrypt 路由设置 Strict-Transport-Security 标头。	
<code>haproxy.router.openshift.io/log-send-hostname</code>	在 Syslog 标头中设置 hostname 字段。使用系统的主机名。如果路由器启用了任何 Ingress API 日志记录方法（如 sidecar 或 Syslog 工具），则默认启用 log-send-hostname 。	
<code>haproxy.router.openshift.io/rewrite-target</code>	在后端中设置请求的重写路径。	
<code>router.openshift.io/cookie-same-site</code>	<p>设置一个值来限制 cookies。值是：</p> <p>Lax : cookies 在访问的站点和第三方站点间进行传输。</p> <p>Strict : cookies 仅限于访问的站点。</p> <p>None : cookies 仅限于指定的站点。</p> <p>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。</p>	

变量	描述	默认的环境变量
haproxy.router.openshift.io/et-forwarded-headers	<p>设置用于处理每个路由的 Forwarded 和 X-Forwarded-For HTTP 标头的策略。值是：</p> <p>Append 附加标头，保留任何现有的标头。这是默认值。</p> <p>replace：设置标头，删除任何现有的标头。</p> <p>Never：不设置标头，而是保留任何现有的标头。</p> <p>if-none：如果没有设置标头，则设置它。</p>	ROUTER_SET_FORWARDED_HEADERS



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	描述
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 在规定时间内没有回答，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。

变量	默认	描述
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。
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	<p>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。</p> <p>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，它被设置为 300s，但 HAProxy 也等待 tcp-request inspect-delay，它被设置为 5s。在这种情况下，整个超时时间将是 300s 加 5s。</p>
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 20.3. 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/

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/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

20.1.8. 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

```

20.1.9. 通过 Ingress 对象创建路由

有些生态系统组件与 **Ingress** 资源集成，但与 **Route** 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
spec:
  rules:
    - host: www.example.com 2
      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。接受的值是 **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。必须在模板文件中定义 TLS 证书详细信息，才能实现默认的边缘路由，并防止生成不安全的路由。

2 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.openshift demos.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. 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
to:
  kind: Service
  name: frontend

```

20.1.10. 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 ❶
    ...
  spec:
    ...
    tls: ❷
      insecureEdgeTerminationPolicy: Redirect
      termination: edge ❸
    ...

```

- ❶ 路由的名称包括 Ingress 对象的名称，后跟一个随机的后缀。
- ❷ 要使用默认证书，路由不应指定 `spec.certificate`。
- ❸ 路由应指定 `edge` 终止策略。

20.1.11. 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: ❶
  - 172.30.0.0/16
  - <second_IP_address>
  ipFamilies: ❷
  - IPv4
  - IPv6
  ipFamilyPolicy: RequireDualStack ❸
  ports:
  - port: 8080
    protocol: TCP
    targetport: 8080
  selector:
    name: <namespace_name>
  sessionAffinity: None
  type: ClusterIP
status:
  loadbalancer: {}

```

- ❶ 在双栈实例中，提供了两个不同的 **clusterIP**。
- ❷ 对于单堆栈实例，输入 **IPv4** 或 **IPv6**。对于双栈实例，请输入 **IPv4** 和 **IPv6**。
- ❸ 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**，请输入以下命令：

```
$ oc get endpointslices
```

其他资源

- [使用 appsDomain 选项指定备选集群域](#)

20.2. 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](#)。

20.2.1. 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。

先决条件

- 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`、`ca.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`。

20.2.2. 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。

先决条件

- 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`。

20.2.3. 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。

先决条件

- 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。

第 21 章 配置集群入口流量

21.1. 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 Load Balancer、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均衡器 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 提供了使用负载均衡器 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。
配置一个 NodePort	在集群中的所有节点上公开某一服务。

21.1.1. 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。

21.2. 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。

21.2.1. 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。

21.2.2. 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。

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

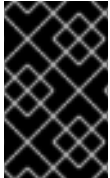
OpenShift Container Platform 在创建 **Service** 对象时使用分配给 **spec.externalIPs[]** 数组的 IP 地址。您不能指定已经被其他服务使用的 IP 地址。

21.2.2.1. 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，则会分配一个带有 **spec.type=LoadBalancer** 的 **Service** 对象。
- 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
```

21.2.2.2. 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。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。

21.2.2.3. 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
  ...

```

21.2.3. 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。

externalIP 小节 1 和 2 的示例配置如下：

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
```

21.2.4. 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"}'
```

21.2.5. 后续步骤

- [为服务外部 IP 配置 ingress 集群流量](#)

21.3. 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。

21.3.1. 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。

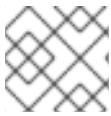
与管理员合作将 Ingress Controller 配置为接受外部请求并根据配置的路由进行代理。

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。

21.3.2. 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。

21.3.3. 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
```

- 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。

21.3.4. 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：

- 登录 OpenShift Container Platform。
- 登录您想公开的服务所在的项目：

```
$ oc project myproject
```

- 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
```

- 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。
 - 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
```

- 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
```

```
...
```

21.3.5. 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。

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: v1
items:
- 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
    status: {}
  kind: List
  metadata:
    resourceVersion: ""
    selfLink: ""
```

- 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
```

21.3.6. 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。

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。



警告

如果您部署 Keepalived Ingress VIP，请不要为 **endpointPublishingStrategy** 参数部署带有值 **HostNetwork** 的非默认 Ingress Controller。这样做可能会导致问题。对于 **endpointPublishingStrategy**，使用 **NodePort** 而不是 **HostNetwork**。

流程

1. 编辑 **router-internal.yaml** 文件：

```
# cat router-internal.yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net> ❶
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    namespaceSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

- ❶ 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。

- 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
```

21.3.7. 其他资源

- Ingress Operator 管理通配符 DNS。如需更多信息，请参阅 [OpenShift Container Platform 中的 Ingress Operator](#)、[在裸机上安装集群](#)和[在 vSphere 上安装集群](#)。

21.4. 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。

21.4.1. 使用负载均衡器使流量进入集群

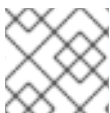
如果不需要具体的外部 IP 地址，您可以配置负载均衡器服务，以便从外部访问 OpenShift Container Platform 集群。

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。

21.4.2. 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。

21.4.3. 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。

21.4.4. 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
...
```

21.4.5. 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为特定的 IP 地址，建议使用 service.beta.kubernetes.io/load-balancer-source-ranges 注解，而不是设置 `loadBalancerSourceRanges` 字段。通过注释，您可以更轻松地迁移到 OpenShift API，后者将在未来的发行版中实施。

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)]>
```

21.5. 使用网络负载均衡器在 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。此方法使用 Network Load Balancer (NLB)，它会将客户端的 IP 地址转发到节点。您可以在新的或现有 AWS 集群上配置 NLB。

21.5.1. 将 Ingress Controller Classic Load Balancer 替换为网络负载均衡器

您可以将使用 Classic 负载均衡器(CLB)的 Ingress Controller 替换为 AWS 上使用网络负载均衡器(NLB)的 Ingress Controller。



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. 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。

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。预计中断的服务器停机时间。

21.5.2. 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](#)。

21.5.3. 在新 AWS 集群上配置 Ingress Controller 网络负载均衡

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。

21.5.4. 其他资源

- [使用自定义网络在 AWS 上安装集群。](#)
- 如需更多信息，请参阅 [AWS 上的网络负载均衡支持](#)。

21.6. 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。

21.6.1. 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。

21.6.2. 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.externalIPs}'
```

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
```

21.6.3. 其他资源

- [为服务配置 ExternalIP](#)

21.7. 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**。

21.7.1. 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。



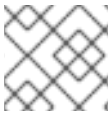
重要

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** 范围内，这意味着 **NodePort** 不太可能与服务的预期端口匹配。例如，端口 **8080** 可能会作为节点上的端口 **31020** 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。

21.7.2. 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。

21.7.3. 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。

21.7.4. 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。OpenShift Container Platform 会自动在 **30000-32767** 范围内选择可用端口。

```
$ oc expose service nodejs-ex --type=NodePort --name=nodejs-ex-nodeport --generator="service/v2"
```

输出示例

```
service/nodejs-ex-nodeport exposed
```

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
```

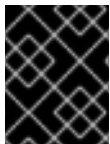

21.7.5. 其他资源

- [配置节点端口服务范围](#)

第 22 章 KUBERNETES NMSTATE

22.1. 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

红帽仅在裸机、IBM Power、IBM Z 和 LinuxONE 上安装生产环境中支持 Kubernetes NMState Operator。



警告

使用 OVN-Kubernetes 时，不支持更改默认网关接口。

在 OpenShift Container Platform 中使用 NMState 之前，必须安装 Kubernetes NMState Operator。

22.1.1. 安装 Kubernetes NMState Operator

您可以使用 web 控制台或 CLI 安装 Kubernetes NMState Operator。

22.1.1.1. 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

- 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。

- 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。

- 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。

22.1.1.2. 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**)。
- 以具有 **cluster-admin** 权限的用户身份登录。

流程

- 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
```

- 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. 创建 **nmstate** operator 实例 :

```
$ cat << EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
EOF
```

验证

- 确认 **nmstate** operator 的部署正在运行 :

```
oc get clusterserviceversion -n openshift-nmstate \
-o custom-columns=Name:.metadata.name,Phase:.status.phase
```

输出示例

```
Name                               Phase
kubernetes-nmstate-operator.4.10.0-202203120157  Succeeded
```

22.2. 观察节点网络状态

节点网络状态是集群中所有节点的网络配置。

22.2.1. 关于 **nmstate**

OpenShift Container Platform 使用 **nmstate** 来报告并配置节点网络的状态。这样就可以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 桥接。

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

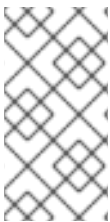
描述节点上请求的网络配置。您可以通过将 **NodeNetworkConfigurationPolicy** 清单应用到集群来更新节点网络配置，包括添加和删除网络接口。

NodeNetworkConfigurationEnactment

报告每个节点上采用的网络策略。

OpenShift Container Platform 支持使用以下 nmstate 接口类型：

- Linux Bridge
- VLAN
- bond
- Ethernet



注意

如果您的 OpenShift Container Platform 集群使用 OVN-Kubernetes 作为默认 Container Network Interface (CNI) 供应商，则无法将 Linux 网桥或绑定附加到主机的默认接口，因为 OVN-Kubernetes 的主机网络拓扑发生了变化。作为临时解决方案，您可以使用连接到主机的二级网络接口，或切换到 OpenShift SDN 默认 CNI 供应商。

22.2.2. 查看节点的网络状态

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。

22.3. 更新节点网络配置

您可以通过将 **NodeNetworkConfigurationPolicy** 清单应用到集群来更新节点网络的配置，如为节点添加或删除接口。



警告

使用 OVN-Kubernetes 时，不支持更改默认网关接口。

22.3.1. 关于 nmstate

OpenShift Container Platform 使用 **nmstate** 来报告并配置节点网络的状态。这样就可以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 桥接。

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。

OpenShift Container Platform 支持使用以下 nmstate 接口类型：

- Linux Bridge
- VLAN
- bond
- Ethernet



注意

如果您的 OpenShift Container Platform 集群使用 OVN-Kubernetes 作为默认 Container Network Interface (CNI) 供应商，则无法将 Linux 网桥或绑定附加到主机的默认接口，因为 OVN-Kubernetes 的主机网络拓扑发生了变化。作为临时解决方案，您可以使用连接到主机的二级网络接口，或切换到 OpenShift SDN 默认 CNI 供应商。

22.3.2. 在节点上创建接口

通过将 **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](#)

22.3.3. 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
```


22.3.4. 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 节点。
- 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。

22.3.5. 不同接口的策略配置示例

22.3.5.1. 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。

22.3.5.2. 示例：VLAN 接口节点网络配置策略

通过将 **NodeNetworkConfigurationPolicy** 清单应用到集群来在集群的节点上创建一个 VLAN 接口。

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。
- 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。

22.3.5.3. 示例：绑定接口节点网络配置策略

通过将 **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**。

22.3.5.4. 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**。

22.3.5.5. 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。

以下示例片断在两个 NIC 间创建一个名为 **bond10** 的绑定和一个名为 **br1** 连接到绑定的 Linux 网桥。

```
#...
  interfaces:
  - name: bond10
    description: Bonding eth2 and eth3 for Linux bridge
    type: bond
    state: up
    link-aggregation:
      port:
      - eth2
      - eth3
  - name: br1
    description: Linux bridge on bond
    type: linux-bridge
    state: up
    bridge:
      port:
      - name: bond10
#...
```

22.3.6. 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。

22.3.6.1. 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 ❶
      prefix-length: 24
  enabled: true
...

```

❶ 使用接口的静态 IP 地址替换这个值。

22.3.6.2. 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
...

```

22.3.6.3. 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

```

```
...
```

22.3.6.4. DNS

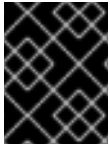
将 DNS 配置设置为修改 `/etc/resolv.conf` 文件。以下片段在主机上设置 DNS 配置。

```

...
interfaces: ❶
  ...
  ipv4:
    ...
    auto-dns: false
    ...
dns-resolver:
  config:
    search:
      - example.com
      - example.org
    server:
      - 8.8.8.8
...

```

- ❶ 您必须配置带有 `auto-dns: false` 的接口，或者您必须在接口上使用静态 IP 配置，以便 Kubernetes NMState 存储自定义 DNS 设置。



重要

在配置 DNS 解析器时，您无法使用 `br-ex`（一个由 OVNKubernetes 管理的 Open vSwitch 网桥）作为接口。

22.3.6.5. 静态路由

以下片段在接口 `eth1` 中配置静态路由和静态 IP。

```

...
interfaces:
- name: eth1
  description: Static routing on eth1
  type: ethernet
  state: up
  ipv4:
    dhcp: false
    address:
      - ip: 192.0.2.251 ❶
        prefix-length: 24
        enabled: true
  routes:
    config:
      - destination: 198.51.100.0/24
        metric: 150
        next-hop-address: 192.0.2.1 ❷

```



```
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。

22.4. 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。

22.4.1. 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 网桥策略应用到一个具有三个 control plane 节点（master）和三个计算（worker）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
```

- 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。

- 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
```

- 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)。在本例中, 端口包含在 **difference** 中, 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 23 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。

23.1. 先决条件

- 查看[集群需要访问的站点](#)中的内容，决定这些站点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

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)。

23.2. 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

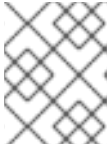
只支持名为 **cluster** 的 **Proxy** 对象，且无法创建额外的代理。

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 `*` 为所有目的地绕过代理。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。

23.3. 删除集群范围代理服务器

`cluster Proxy` 对象不能被删除。要删除一个集群中的代理，从 Proxy 对象中删除所有 `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](#)

第 24 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，其中包含您定义的额外 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。

24.1. 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: example.com 3
additionalTrustBundle: | 4
  -----BEGIN CERTIFICATE-----
  <MY_TRUSTED_CA_CERT>
  -----END CERTIFICATE-----
...

```

- 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。在域前面加上 **.** 以仅匹配子域。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。
- 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。



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。

24.2. 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

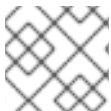
```

```

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。

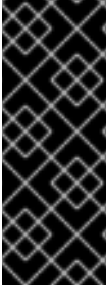
在域前面加 **.** 来仅匹配子域。例如：**.y.com** 匹配 **x.y.com**，但不匹配 **y.com**。使用 ***** 为所有目的地绕过代理。如果您扩展了未包含在安装配置中 **networking.machineNetwork[].cidr** 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

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。

24.3. 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
```

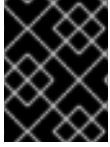
```
    readOnly: true
  volumes:
  - name: trusted-ca
    configMap:
      name: trusted-ca
      items:
      - key: ca-bundle.crt 1
        path: tls-ca-bundle.pem 2
```

- 1** **ca-bundle.crt** 需要作为 ConfigMap 键。
- 2** **TLS-ca-bundle.pem** 需要作为 ConfigMap 路径。

第 25 章 RHOSP 负载均衡

25.1. 使用带有 KURYR SDN 的 OCTAVIA OVN 负载均衡器供应商驱动

如果您的 OpenShift Container Platform 集群使用 Kuryr，并安装在稍后升级到 RHOSP 16 的 Red Hat OpenStack Platform (RHOSP) 13 云上，您可以将其配置为使用 Octavia OVN 供应商驱动程序。



重要

在更改供应商驱动程序后，Kuryr 会替换现有的负载均衡器。这个过程会产生一些停机时间。

先决条件

- 安装 RHOSP CLI **openstack**。
- 安装 OpenShift Container Platform CLI **oc**。
- 验证 RHOSP 上 Octavia OVN 驱动程序是否启用。

提示

要查看可用 Octavia 驱动程序列表，请在命令行中输入 **openstack loadbalancer provider list**。命令的输出中会显示 **ovn** 驱动。

流程

把 Octavia Amphora 供应商驱动改为 Octavia OVN:

1. 打开 **kuryr-config** ConfigMap。在命令行中运行：

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

2. 在 ConfigMap 中，删除包含 **kuryr-octavia-provider: default** 的行。例如：

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: default 1
...
```

- 1** 删除这一行。集群将会重新生成，并带有 **ovn** 值。

等待 Cluster Network Operator 检测修改并重新部署 **kuryr-controller** 和 **kuryr-cni** pod。这可能需要几分钟。

3. 验证 **kuryr-config** ConfigMap 注解是否带有 **ovn** 作为其值。在命令行中运行：

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

ovn 供应商值会在输出中显示：

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: ovn
...
```

4. 验证 RHOSP 是否已重新创建其负载均衡器。

a. 在命令行中运行：

```
$ openstack loadbalancer list | grep amphora
```

此时会显示一个 Amphora 负载均衡器。例如：

```
a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer |
84c99c906edd475ba19478a9a6690efd | 172.30.0.1 | ACTIVE | amphora
```

b. 输入以下内容查找 **ovn** 负载均衡器：

```
$ openstack loadbalancer list | grep ovn
```

显示 **ovn** 类型的负载均衡器。例如：

```
2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE | ovn
0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd |
84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE | ovn
f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.152.27 | ACTIVE | ovn
```

25.2. 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。

如果您的集群使用 Kuryr, Cluster Network Operator 会在部署时创建一个内部 Octavia 负载均衡器。您可以使用此负载均衡器进行应用程序网络扩展。

如果您的集群没有使用 Kuryr, 则需要创建自己的 Octavia 负载均衡器将其用于应用程序网络扩展。

25.2.1. 使用 Octavia 扩展集群

如果要使用多个 API 负载均衡器, 或者集群没有使用 Kuryr, 请创建一个 Octavia 负载均衡器, 然后配置集群使用它。

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。



注意

如果 Kuryr 使用 Octavia Amphora 驱动程序，则所有流量都通过单个 Amphora 虚拟机 (VM) 路由。

您可以重复这个过程来创建其他负载均衡器，这样可降低瓶颈。

25.2.2. 通过 Octavia 扩展使用 Kuryr 的集群

如果您的集群使用 Kuryr，将集群的 API 浮动 IP 地址与预先存在的 Octavia 负载均衡器相关联。

先决条件

- OpenShift Container Platform 集群使用 Kuryr。
- Octavia 包括在您的 Red Hat OpenStack Platform (RHOSP) 部署中。

流程

1. 可选：在命令行中，为了重新使用集群 API 浮动 IP 地址取消设置它：

```
$ openstack floating ip unset $API_FIP
```

2. 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 ${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
```

您的集群现在使用 Octavia 进行负载平衡。



注意

如果 Kuryr 使用 Octavia Amphora 驱动程序，则所有流量都通过单个 Amphora 虚拟机 (VM) 路由。

您可以重复这个过程来创建其他负载均衡器，这样可降低瓶颈。

25.3. 使用 RHOSP OCTAVIA 为入站流量扩展

您可以使用 Octavia 负载均衡器来扩展使用 Kuryr 的集群中的 Ingress 控制器。

先决条件

- OpenShift Container Platform 集群使用 Kuryr。
- Octavia 可用于您的 RHOSP 部署。

流程

1. 要复制当前的内部路由器服务，在命令行中输入：

```
$ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
```

- 在 `external_router.yaml` 文件中，将 `metadata.name` 和 `spec.type` 的值改为 **LoadBalancer**。

路由器文件示例

```

apiVersion: v1
kind: Service
metadata:
  labels:
    ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
  name: router-external-default 1
  namespace: openshift-ingress
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
    - name: metrics
      port: 1936
      protocol: TCP
      targetPort: 1936
  selector:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  sessionAffinity: None
  type: LoadBalancer 2

```

- 1 确保此值具有描述性，如 **router-external-default**。
- 2 确定这个值是 **LoadBalancer**。



注意

您可以删除与负载均衡相关的时间戳和其他信息。

1. 在命令行中，从 `external_router.yaml` 文件创建服务：

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

2. 验证服务的外部 IP 地址是否与与负载均衡器关联的 IP 地址相同：

- a. 在命令行中检索服务的外部 IP 地址：

```
$ oc -n openshift-ingress get svc
```

输出示例

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
router-external-default	LoadBalancer	172.30.235.33	10.46.22.161	

```
80:30112/TCP,443:32359/TCP,1936:30317/TCP 3m38s
router-internal-default ClusterIP 172.30.115.123 <none>
80/TCP,443/TCP,1936/TCP 22h
```

- b. 检索负载均衡器的 IP 地址：

```
$ openstack loadbalancer list | grep router-external
```

输出示例

```
| 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-default |
66f3816acf1b431691b8d132cc9d793c | 172.30.235.33 | ACTIVE | octavia |
```

- c. 验证您在前面的步骤中获取的地址是否在浮动 IP 列表中相互关联：

```
$ openstack floating ip list | grep 172.30.235.33
```

输出示例

```
| e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-
806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb |
66f3816acf1b431691b8d132cc9d793c |
```

现在，您可以使用 **EXTERNAL-IP** 值作为新的入口地址。



注意

如果 Kuryr 使用 Octavia Amphora 驱动程序，则所有流量都通过单个 Amphora 虚拟机 (VM) 路由。

您可以重复这个过程来创建其他负载均衡器，这样可降低瓶颈。

25.4. 配置外部负载均衡器

您可以在 Red Hat OpenStack Platform (RHOSP) 上配置 OpenShift Container Platform 集群，使其使用外部负载均衡器来代替默认负载均衡器。

先决条件

- 在负载均衡器中，系统任意用户使用端口 6443、443 和 80 的 TCP。
- 在每个 control plane 节点之间负载平衡 API 端口 6443。
- 在所有计算节点之间负载平衡应用程序端口 443 和 80。
- 在负载均衡器中，用于为节点提供 ignition 启动配置的端口 22623 不会在集群外公开。
- 您的负载均衡器必须能够访问集群中的每台机器。允许此访问的方法包括：
 - 将负载均衡器附加到集群的机器子网。
 - 将浮动 IP 地址附加到使用负载均衡器的机器。

流程

1. 在端口 6443、443 和 80 中启用从负载均衡器访问集群的功能。
例如，请注意此 HAProxy 配置：

HAProxy 配置示例

```
...
listen my-cluster-api-6443
  bind 0.0.0.0:6443
  mode tcp
  balance roundrobin
  server my-cluster-master-2 192.0.2.2:6443 check
  server my-cluster-master-0 192.0.2.3:6443 check
  server my-cluster-master-1 192.0.2.1:6443 check
listen my-cluster-apps-443
  bind 0.0.0.0:443
  mode tcp
  balance roundrobin
  server my-cluster-worker-0 192.0.2.6:443 check
  server my-cluster-worker-1 192.0.2.5:443 check
  server my-cluster-worker-2 192.0.2.4:443 check
listen my-cluster-apps-80
  bind 0.0.0.0:80
  mode tcp
  balance roundrobin
  server my-cluster-worker-0 192.0.2.7:80 check
  server my-cluster-worker-1 192.0.2.9:80 check
  server my-cluster-worker-2 192.0.2.8:80 check
```

2. 在集群 API 的 DNS 服务器中添加记录，并在负载均衡器上应用记录。例如：

```
<load_balancer_ip_address> api.<cluster_name>.<base_domain>
<load_balancer_ip_address> apps.<cluster_name>.<base_domain>
```

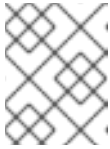
3. 在命令行中，使用 **curl** 验证外部负载均衡器和 DNS 配置是否正常运行。
 - a. 验证集群 API 是否可访问：

```
$ 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. 验证集群应用程序是否可以访问：



注意

您还可以在 Web 浏览器中打开 OpenShift Container Platform 控制台来验证应用程序的可访问性。

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

如果配置正确，您会收到 HTTP 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/JuLJMeoKNXIfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; 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
```

第 26 章 使用 METALLB 进行负载均衡

26.1. 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。

您可以配置 MetalLB，以便使用第 2 层协议公告 IP 地址。通过第 2 层，MetalLB 提供了容错的外部 IP 地址。

您可以配置 MetalLB，以便使用 BGP 协议公告 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 支持为某些 IP 地址和 BGP 提供第 2 层用于其他 IP 地址。

26.1.1. 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 gossip 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。

26.1.2. 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。

AddressPool

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。当您在集群中添加 **AddressPool** 自定义资源时，MetalLB Operator 会配置 MetalLB，以便它能够从池中分配 IP 地址。地址池包含 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。地址池需要名称。文档使用 **doc-example**、**doc-example-reserved** 和 **doc-example-ipv6** 等名称。地址池指定 MetalLB 是否可以自动从池中分配 IP 地址，或者是否为按名称显式指定池的服务保留 IP 地址。一个地址池指定 MetalLB 是否使用第 2 层协议公告 IP 地址，还是使用 BGP 协议。

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 将服务负载均衡器 IP 地址的路由公告给一个或多个 BGP 对等

点。服务负载均衡器 IP 地址通过 **AddressPool** 自定义资源来指定，这些资源将 **protocol** 字段设置为 **bgp**。

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。

将 **MetalLB** 自定义资源添加到集群后，Operator 将部署 MetalLB、MetalLB 软件组件、**controller** 和 **speaker**。

Operator 包含为 **AddressPool** 和 **BGPPeer** 自定义资源验证 Webhook。地址池自定义资源的 Webhook 执行以下检查：

- 地址池名称必须是唯一的。
- IP 地址范围不与现有地址池重叠。
- 如果地址池包含 **bgpAdvertisement** 字段，则 **protocol** 字段必须设置为 **bgp**。

BGP peer 自定义资源的 webhook 执行以下检查：

- 如果 BGP peer 名称与现有 peer 匹配，则 peer 的 IP 地址必须是唯一的。
- 如果指定了 **keepaliveTime** 字段，则必须指定 **holdTime** 字段，并且 keep-alive 持续时间必须小于保留的时间。
- 所有 BGP 对等点的 **myASN** 字段必须相同。

26.1.3. 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 **MetalLB** 自定义资源和 **AddressPool** 自定义资源的更改。

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。

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 **控制器** 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 BGP 模式，**控制器** 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 地址的请求通过 **speaker** 路由到宣布 IP 地址的节点。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。

26.1.4. 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

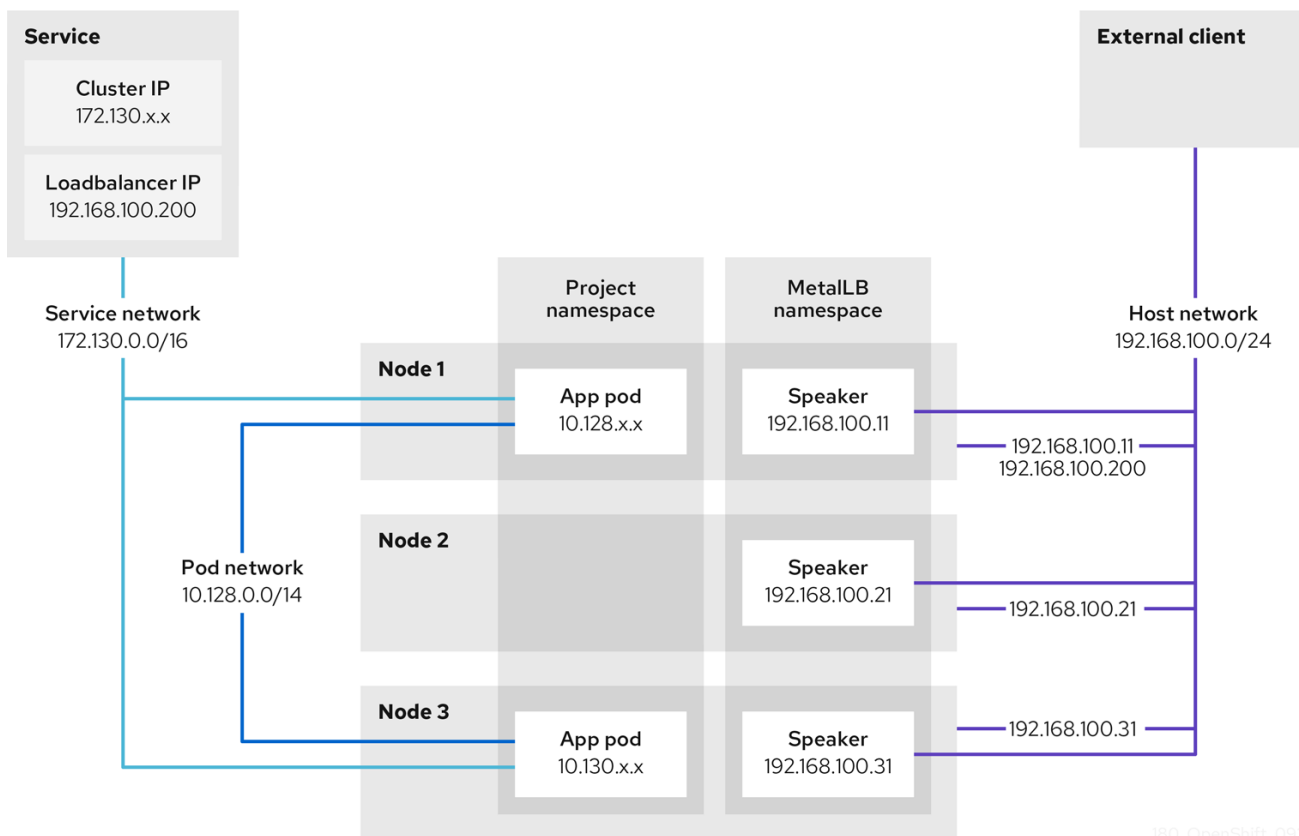
在第 2 层模式中，MetalLB 依赖于 ARP 和 NDP。这些协议在特定子网中实施本地地址解析。在这种情况下，客户端必须能够访问由 MetalLB 分配的 VIP，它与节点位于同一个子网中，以便 MetalLB 正常工作。

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 层实施负载均衡器。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。



180_OpenShift_0921

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。该服务也有一个外部 IP 地址，MetalLB 分配到该服务 **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 都是一个主机网络的 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 必须与该服务的端点位于同一个节点上，端点必须处于 **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。

26.1.5. BGP 模式的 MetalLB 概念

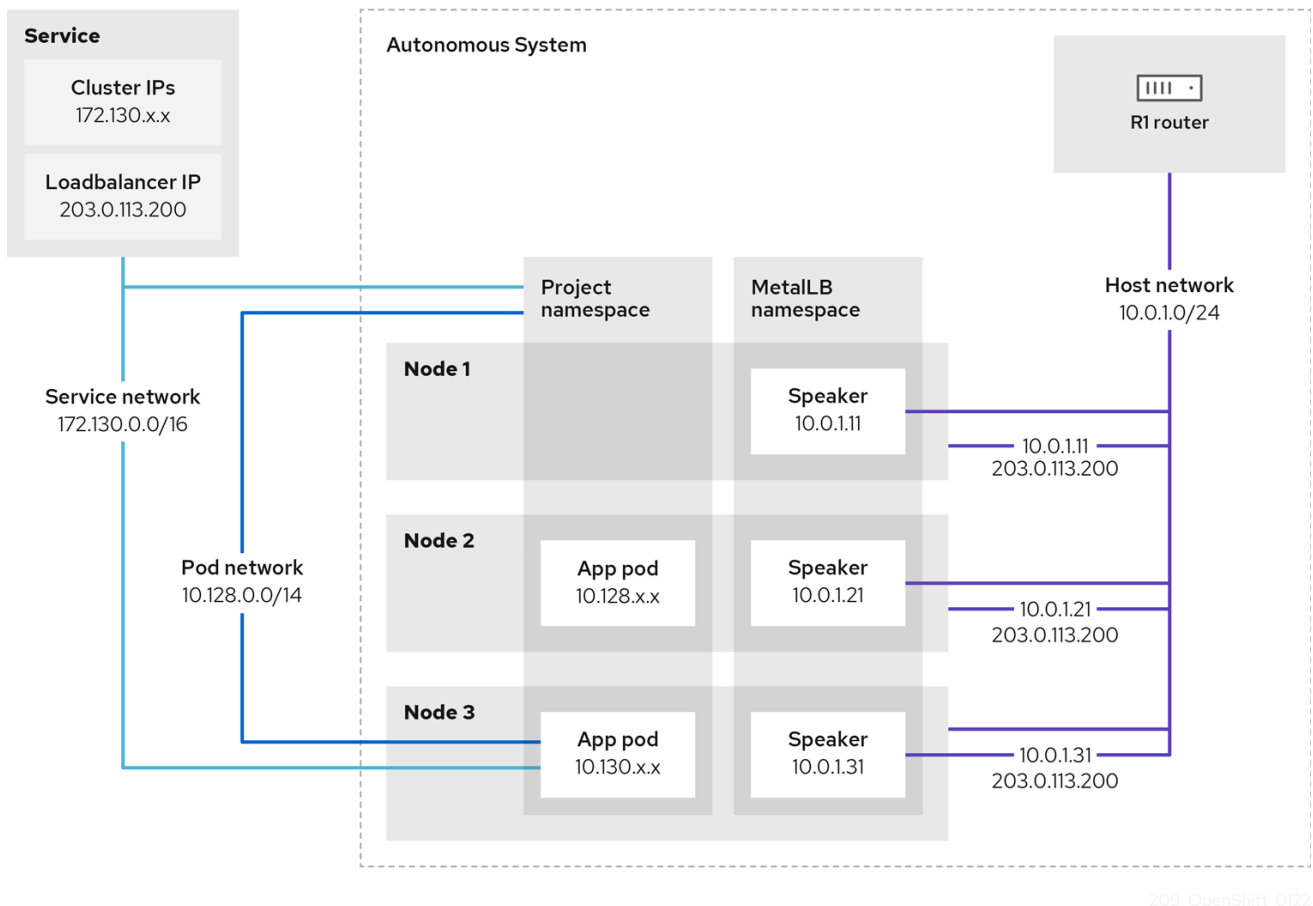
在 BGP 模式中，每个 **speaker** pod 都会向每个 BGP 对等点公告一个服务的负载均衡器 IP 地址。BGP 对等点是配置为使用 BGP 协议的网络路由器。当路由器收到负载均衡器 IP 地址的流量时，路由器会选择带有公告 IP 地址的 **speaker** pod 的节点。路由器将流量发送到该节点。在流量进入节点后，CNI 网络供应商的服务代理会将流量分发到该服务的所有 pod。

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 26.1. BGP 模式的 MetalLB 拓扑图



209_OpenShift_0122

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 都是一个主机网络的 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** 负载均衡器 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。

26.1.6. 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。

26.1.7. 限制和限制

26.1.7.1. 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

OpenShift SDN 和 OVN-Kubernetes 网络供应商支持 MetalLB 和 MetalLB。

26.1.7.2. 第 2 层模式的限制

26.1.7.2.1. 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。

26.1.7.2.2. 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。

26.1.7.3. BGP 模式限制

26.1.7.3.1. 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 的负载均衡。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。

26.1.7.3.2. 社区被指定为 16 位值

社区作为地址池自定义资源的一部分指定，以冒号分隔的 16 位值的形式指定。例如，要指定，使用知名 **NO_ADVERTISE** 社区属性公告负载均衡器 IP 地址，请使用如下表示法：

```
apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-no-advertise
  namespace: metallb-system
spec:
  protocol: bgp
  addresses:
    - 192.168.1.100-192.168.1.255
  bgpAdvertisements:
    - communities:
      - 65535:65282
```

有些社区具有仅能指定 16 位值的限制，这与社区支持的 MetalLB 实施不同，后者支持一个 **bgp-communities** 项和用于 BGP 社区的可读名称。

26.1.7.3.3. 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。

26.1.8. 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](#)

26.2. 安装 METALLB OPERATOR

作为集群管理员，您可以添加 MetallB Operator，以便 Operator 可以管理集群中的 MetalLB 实例的生命周期。

安装过程使用 **metallb-system** 命名空间。您可以安装 Operator 并在不同的命名空间中配置自定义资源。Operator 在安装 Operator 的同一命名空间中启动 MetalLB。

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。

26.2.1. 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。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 可选：为 MetalLB Operator 创建所需的命名空间：



注意

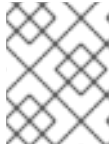
您可以选择在这个阶段创建命名空间，或者在启动 MetalLB Operator 安装时创建命名空间。在 **Installed Namespace** 列表中，您可以创建项目。

- a. 进行 **Administration** → **Namespaces**，点 **Create Namespace**。
 - b. 在 **Name** 字段中输入 **metallb-system**，然后点 **Create**。
3. 安装 MetalLB Operator：
 - a. 在 OpenShift Container Platform Web 控制台中，点击 **Operators** → **OperatorHub**。
 - b. 在 **Filter by keyword** 字段中输入 **metallb** 以查找 MetalLB Operator，然后点 **Install**。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**。
 - c. 在 **Install Operator** 页面中，选择**集群上的特定命名空间**。选择在上一节中创建的命名空间，或选择创建 **metallb-system** 项目，然后点 **Install**。

验证

验证 MetalLB Operator 是否已成功安装：

1. 进入到 **Operators → Installed Operators** 页面。
2. 确保 **metallb-system** 项目中列出的 **MetalLB Operator** 的 **Status** 为 **Succeeded**。



注意

在安装过程中，Operator 可能会显示 **Failed** 状态。如果安装后安装成功并显示 **Succeeded** 信息，您可以忽略 **Failed** 信息。

3. 如果 Operator 安装不成功，您可以进一步排除故障：
 - a. 导航到 **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。
 - b. 进入到 **Workloads → Pods** 页面，在 **metallb-system** 项目中检查 pod 的日志。

26.2.2. 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。使用 **oc** 命令来创建或更新一个订阅对象。

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. 确认 MetalLB Operator 可用：

```
$ oc get packagemanifests -n openshift-marketplace metallb-operator
```

输出示例

```
NAME          CATALOG          AGE
metallb-operator  Red Hat Operators  9h
```

2. 创建 **metallb-system** 命名空间：

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

3. 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"
```

4. 在命名空间中创建一个 Operator 组自定义资源：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
spec:
  targetNamespaces:
  - metallb-system
EOF
```

5. 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
```

6. 订阅 MetalLB Operator。

- a. 运行以下命令以获取 OpenShift Container Platform 的主版本和次版本。您可以使用这些值来设置下一步中的 **channel** 值。

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
grep -o '[0-9]*[.][0-9]*' | head -1)
```

- b. 要为 Operator 创建订阅自定义资源，请输入以下命令：

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metallb-operator-sub
  namespace: metallb-system
spec:
  channel: "${OC_VERSION}"
  name: metallb-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

7. 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.4.10.0-nnnnnnnnnnnn Automatic true

```

- 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.4.10.0-nnnnnnnnnnnn Succeeded

```

26.2.3. 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

- 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。

- 检查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

```

- 检查 speaker 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。

26.2.3.1. 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: <.>
    node-role.kubernetes.io/worker: ""
  speakerTolerations: <.>
    - key: "Example"
      operator: "Exists"
      effect: "NoExecute"
```

<.> 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。<.> 在这个示例配置中，附加此容限的 pod 会容限使用 **operator** 的，与 **key** 的值和 **effect** 值匹配的污点。

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。

其他资源

- 如需有关节点选择器的更多信息，请参阅[使用节点选择器将 pod 放置到特定的节点上](#)。
- 如需有关污点和容限的更多信息，请参阅[了解污点和容限](#)。

26.2.4. 后续步骤

- [配置 MetalLB 地址池](#)

26.3. 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。

26.3.1. 关于地址池自定义资源

下表中描述了地址池自定义资源的字段。

表 26.1. MetalLB 地址池自定义资源

字段	类型	描述
<code>metadata.name</code>	字符串	指定地址池的名称。添加服务时，您可以在 <code>metallb.universe.tf/address-pool</code> 注解中指定这个池名称，以从特定池中选择 IP 地址。在整个文档中使用名称 <code>doc-example</code> , <code>silver</code> , 和 <code>gold</code> 。
<code>metadata.name space</code>	字符串	指定地址池的命名空间。指定 MetalLB Operator 使用的同一命名空间。
<code>spec.protocol</code>	字符串	指定向对等节点发布负载均衡器 IP 地址的协议。指定 <code>layer2</code> 或 <code>bgp</code> 。
<code>spec.autoAssign</code>	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 <code>metallb.universe.tf/address-pool</code> 注解从这个池中明确请求 IP 地址，请指定 <code>false</code> 。默认值为 <code>true</code> 。
<code>spec.addresses</code>	数组	指定分配给服务的 MetalLB 的 IP 地址列表。您可以在单个池中指定多个范围。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。
<code>spec.bgpAdvertisements</code>	object	可选：默认情况下，BGP 模式向配置的对等点公告每个分配的负载均衡器 IP 地址，且没有额外的 BGP 属性。对等路由器收到每个服务 IP 地址的一个 <code>/32</code> 路由，BGP 本地首选项设置为零，没有 BGP 社区。使用此字段创建自定义广告。

`bgpAdvertisements` 对象的字段在下表中定义：

表 26.2. BGP 公告配置

字段	类型	描述
----	----	----

字段	类型	描述
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。
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。
community	数组	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： <ul style="list-style-type: none"> ● NO_EXPORT: 65535:65281 ● NO_ADVERTISE: 65535:65282 ● NO_EXPORT_SUBCONFED: 65535:65283
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。

26.3.2. 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. 创建一个包含类似以下示例内容的文件，如 **addresspool.yaml**：

```
apiVersion: metallb.io/v1alpha1
kind: AddressPool
metadata:
  namespace: metallb-system
  name: doc-example
spec:
  protocol: layer2
  addresses:
    - 203.0.113.1-203.0.113.10
    - 203.0.113.65-203.0.113.75
```

2. 为地址池应用配置：

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

验证

- 查看地址池：

```
$ oc describe -n metallb-system addresspool doc-example
```

输出示例

```
Name:      doc-example
Namespace: metallb-system
Labels:    <none>
Annotations: <none>
API Version: metallb.io/v1alpha1
Kind:      AddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign: true
  Protocol:    layer2
  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。

26.3.3. 地址池配置示例

26.3.3.1. 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: AddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  protocol: layer2
  addresses:
    - 192.168.100.0/24
    - 192.168.200.0/24
    - 192.168.255.1-192.168.255.5
```

26.3.3.2. 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: AddressPool
metadata:
```

```

name: doc-example-reserved
namespace: metallb-system
spec:
  protocol: layer2
  addresses:
  - 10.0.100.0/28
  autoAssign: false

```

26.3.3.3. 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: AddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  protocol: layer2
  addresses:
  - 10.0.100.0/28
  - 2002:2:2::1-2002:2:2::100

```

26.3.3.4. 示例：使用 BGP 模式的简单地址池

对于 BGP 模式，您必须将 **protocol** 字段设置为 **bgp**。其他地址池自定义资源字段（如 **autoAssign**）也适用于 BGP 模式。

在以下示例中，对等的 BGP 路由器接收一个 **203.0.113.200/32** 路由，以及一个 **fc00:f853:ccd:e799::1/128** 路由，每个 load-balancer IP 地址分配给服务。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。

```

apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-bgp
  namespace: metallb-system
spec:
  protocol: bgp
  addresses:
  - 203.0.113.200/30
  - fc00:f853:ccd:e799::/124

```

26.3.3.5. 示例：带有自定义公告的 BGP 模式

您可以指定复杂的自定义公告。

```

apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-bgp-adv

```

```

namespace: metallb-system
spec:
  protocol: bgp
  addresses:
  - 203.0.113.200/30
  - fc00:f853:ccd:e799::/124
  bgpAdvertisements:
  - communities:
    - 65535:65282
    aggregationLength: 32
    localPref: 100
  - communities:
    - 8000:800
    aggregationLength: 30
    aggregationLengthV6: 124

```

在前面的示例中，MetalLB 分配 IP 地址到 **203.0.113.200** 和 **203.0.113.203** 之间的范围中的负载均衡服务，并在 **fc00:f853:ccd:e799::0** 和 **fc00:f853:ccd:e799::f** 之间。

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**，并且社区设置为知名 **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。

26.3.4. 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](#)。

26.4. 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。

26.4.1. 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 26.3. MetalLB BGP peer 自定义资源

字段	类型	描述
<code>metadata.name</code>	字符串	指定 BGP peer 自定义资源的名称。
<code>metadata.name space</code>	字符串	指定 BGP peer 自定义资源的命名空间。
<code>spec.myASN</code>	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。范围为 0 到 65535 。
<code>spec.peerASN</code>	整数	为 BGP 会话的远程端指定 Autonomous System 号。范围为 0 到 65535 。
<code>spec.peerAddress</code>	字符串	指定建立 BGP 会话的对等点的 IP 地址。
<code>spec.sourceAddress</code>	字符串	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。
<code>spec.peerPort</code>	整数	可选：指定用来建立 BGP 会话的对等端口。范围为 0 到 16384 。
<code>spec.holdTime</code>	字符串	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。
<code>spec.keepaliveTime</code>	字符串	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。
<code>spec.routerID</code>	字符串	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。
<code>spec.password</code>	字符串	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。
<code>spec.bfdProfile</code>	字符串	可选：指定 BFD 配置集的名称。
<code>spec.nodeSelectors</code>	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。
<code>spec.ebgpMultiHop</code>	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 <i>外部 BGP</i> 。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。

26.4.2. 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。
- 配置为 **spec.protocol** 字段指定 **bgp** 的 MetalLB 地址池。

流程

1. 创建一个文件，如 **bgppeer.yaml**，其内容类似以下示例：

```
apiVersion: metallb.io/v1beta1
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
```

其他资源

- [示例：使用 BGP 模式的简单地址池](#)
- 配置为 **spec.protocol** 字段指定 **bgp** 的 MetalLB 地址池。

26.4.3. BGP 对等配置示例

26.4.3.1. 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/v1beta1
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]
```

26.4.3.2. 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/v1beta1
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](#)。

26.4.3.3. 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/v1beta1
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/v1beta1
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500
```

其他资源

- 将服务配置为使用 MetalLB

26.5. 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。

26.5.1. 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 26.4. BFD 配置集自定义资源

字段	类型	描述
<code>metadata.name</code>	字符串	指定 BFD 配置集自定义资源的名称。
<code>metadata.name space</code>	字符串	指定 BFD 配置集自定义资源的命名空间。
<code>spec.detectMultiplier</code>	整数	<p>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。</p> <p>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。</p> <p>范围为 2 到 255。默认值为 3。</p>
<code>spec.echoMode</code>	布尔值	<p>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。</p> <p>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。</p>
<code>spec.echoInterval</code>	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 。
<code>spec.minimumTtl</code>	整数	<p>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。</p> <p>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。</p> <p>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。</p>

字段	类型	描述
spec.passiveMode	布尔值	<p>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。</p> <p>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。</p> <p>默认值为 false，并将会话标记为 active。</p>
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。
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。

26.5.2. 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
```

26.5.3. 后续步骤

- 将 BGP peer 配置为使用 BFD 配置集。

26.6. 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。

26.6.1. 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
```

26.6.2. 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。

26.6.3. 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

```

26.6.4. 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" ⑤
spec:
  ports:
    - name: https
      port: 443 ⑥
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ⑦
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ⑧

```

① ⑤ 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 ⑥ 为服务指定不同的端口号。

③ ⑦ 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 ⑧ 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。

26.6.5. 使用 MetalLB 配置服务

您可以将负载均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 <.>
Selector:      app=service_name
Type:          LoadBalancer <.>
IP Family Policy:  SingleStack
IP Families:   IPv4
IP:            10.105.237.254
IPs:           10.105.237.254
LoadBalancer Ingress:  192.168.100.5 <.>
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: <.>
  Type Reason      Age          From          Message
  ---- -
  Normal nodeAssigned 32m (x2 over 32m) metallb-speaker announcing from node "
  <node_name>"
```

<.> 注解存在，如果您从特定池请求 IP 地址。<.> 服务类型必须指示 **LoadBalancer**。<.> 如果服务被正确分配，则 `load-balancer ingress` 字段指示外部 IP 地址。<.> `events` 字段表示分配给声明外部 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。

26.7. 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。

26.7.1. 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** 作为理解是 **metallb** CR 已创建，此处您要更改日志级别。

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-xtsrxr\",\"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:\":\"\"},\"f:endpoints.kubernetes.io/last-change-trigger-time\":{\"f:generateName\":{\"f:labels\":{\"f:\":\"\"},\"f:app\":{\"f:endpointslice.kubernetes.io/managed-by\":{\"f:kubernetes.io/service-name\":{\"f:ownerReferences\":{\"f:\":\"\"},\"k\":{\"uid\":\"0518eed3-6152-42be-b566-0bd00a60faf8\"}}}}},\"f:ports\":{\"f: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

```

26.7.1.1. FRRouting(FRR)日志级别

下表描述了 FRR 日志记录级别。

表 26.5. 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。

26.7.2. 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 app.kubernetes.io/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 -- vtsh -c "show running-config"
```

输出示例

```
Building configuration...
```

```
Current configuration:
!
frf version 7.5.1_git
frf defaults traditional
hostname some-hostname
log file /etc/frf/frf.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 ④
  neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full ⑤
  neighbor 10.0.2.4 timers 5 15
  !
  address-family ipv4 unicast
    network 203.0.113.200/30 ⑥
    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 ⑦
    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 ⑧
    transmit-interval 35
    receive-interval 35
    passive-mode
    echo-mode
    echo-interval 35
    minimum-ttl 10
```

```
!
!
end
```

<.> **router bgp** 部分表示 MetalLB 的 ASN。<.> 确认存在 **neighbor <ip-address> remote-as <peer-ASN>** 行，用于您添加的每个 BGP peer 自定义资源。<.> 如果配置了 BFD，请确认 BFD 配置集与正确的 BGP peer 关联，并确认 BFD 配置集会出现在命令输出中。<.> 确认 **network <ip-address-range>** 行与您在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 3
10.0.2.4      4     64500     0       0     0  0  0 never      Active      0 4

Total number of neighbors 2
```

1 1 3 确认输出包含您添加的每个 BGP peer 自定义资源的行。

2 4 2 4 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 <.>
  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
```

<.> 确认输出中包含 BGP peer 的 IP 地址。

26.7.3. 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 app.kubernetes.io/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 -- vttysh -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。

26.7.4. BGP 和 BFD 的 MetalLB 指标

OpenShift Container Platform 捕获以下与 MetalLB 和 BGP peer 和 BFD 配置集相关的指标：

- **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 peer 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 echo 数据包的数量。
- **metallb_bfd_echo_packet_output** 计算发送到每个 BFD 对等点的 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。
- **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。
- **metallb_bgp_session_up** 表示连接状态与 BGP peer。**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 BGP peer 的 BGP **更新**信息的数量。

其他资源

- 有关使用监控仪表板的信息，请参阅 [查询指标](#)。

26.7.5. 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 **vttysh** 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 27 章 将二级接口指标与网络附加关联

27.1. 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。

27.1.1. 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。

27.1.2. 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="namespace",network_name="namespace/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。

使用类似以下的 `promql` 查询时，可以获取包含这个值的新指标，以及从 `k8s.v1.cni.cncf.io/networks-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)
```

第 28 章 NETWORK OBSERVABILITY (网络可观察性)

28.1. NETWORK OBSERVABILITY OPERATOR 发行注记

Network Observability Operator 可让管理员观察和分析 OpenShift Container Platform 集群的网络流量流。

本发行注记介绍了 OpenShift Container Platform 中 Network Observability Operator 的开发。

有关 Network Observability Operator 的概述，请参阅[关于 Network Observability Operator](#)。

28.1.1. Network Observability Operator 1.3.0

以下公告可用于 Network Observability Operator 1.3.0：

- [RHSA-2023:3905 Network Observability Operator 1.3.0](#)

28.1.1.1. 频道弃用

您必须将频道从 **v1.0.x** 切换到 **stable**，以接收将来的 Operator 更新。**v1.0.x** 频道已弃用，计划在以后的发行版本中删除。

28.1.1.2. 新功能及功能增强

28.1.1.2.1. Network Observability 中的多租户

- 系统管理员可以将单独的用户访问或组访问权限限制为存储在 Loki 中的流。如需更多信息，请参阅[Network Observability 中的多租户](#)。

28.1.1.2.2. 基于流的指标仪表盘

- 此发行版本添加了一个新的仪表盘，它概述了 OpenShift Container Platform 集群中的网络流。如需更多信息，请参阅[Network Observability 指标](#)。

28.1.1.2.3. 使用 must-gather 工具进行故障排除

- 有关 Network Observability Operator 的信息现在可以包含在 must-gather 数据中以进行故障排除。如需更多信息，请参阅[Network Observability must-gather](#)。

28.1.1.2.4. 现在支持多个构架

- Network Observability Operator 现在可在 amd64、ppc64le 或 arm64 架构上运行。在以前的版本中，它只在 amd64 上运行。

28.1.1.3. 已弃用的功能

28.1.1.3.1. 弃用的配置参数设置

Network Observability Operator 1.3 发行版本弃用了 **spec.Loki.authToken HOST** 设置。使用 Loki Operator 时，现在必须使用 **FORWARD** 设置。

28.1.1.4. 程序错误修复

- 在以前的版本中，当通过 CLI 安装 Operator 时，Cluster Monitoring Operator 所需的 **Role** 和 **RoleBinding** 不会按预期安装。从 Web 控制台安装 Operator 时，不会出现这个问题。现在，安装 Operator 的任何方法都会安装所需的 **Role** 和 **RoleBinding**。([NETOBSERV-1003](#))
- 自版本 1.2 起，Network Observability Operator 可以在流集合出现问题时引发警报。在以前的版本中，由于一个程序错误，用于禁用警报的相关配置，**spec.processor.metrics.disableAlerts** 无法正常工作，有时无效。现在，此配置已被修复，可以禁用警报。([NETOBSERV-976](#))
- 在以前的版本中，当 Network Observability 被配置为 **spec.loki.authToken** 为 **DISABLED** 时，只有 **kubeadmin** 集群管理员才能查看网络流。其他类型的集群管理员收到授权失败。现在，任何集群管理员都可以查看网络流。([NETOBSERV-972](#))
- 在以前的版本中，一个 bug 会阻止用户将 **spec.consolePlugin.portNaming.enable** 设置为 **false**。现在，此设置可以设置为 **false** 来禁用端口到服务名称转换。([NETOBSERV-971](#))
- 在以前的版本中，Cluster Monitoring Operator (Prometheus) 不会收集由 console 插件公开的指标，因为配置不正确。现在，配置已被修复，控制台插件指标可以被正确收集并从 OpenShift Container Platform Web 控制台访问。([NETOBSERV-765](#))
- 在以前的版本中，当在 **FlowCollector** 中将 **processor.metrics.tls** 设置为 **AUTO** 时，**flowlogs-pipeline servicemonitor** 不会适应适当的 TLS 方案，且指标在 web 控制台中不可见。现在，这个问题已针对 AUTO 模式解决。([NETOBSERV-1070](#))
- 在以前的版本中，证书配置（如 Kafka 和 Loki）不允许指定 namespace 字段，这意味着证书必须位于部署 Network Observability 的同一命名空间中。另外，当在 TLS/mTLS 中使用 Kafka 时，用户必须手动将证书复制到部署 eBPF 代理 pod 的特权命名空间，并手动管理证书更新，如证书轮转时。现在，通过在 **FlowCollector** 资源中为证书添加 namespace 字段来简化 Network Observability 设置。现在，用户可以在不同的命名空间中安装 Loki 或 Kafka，而无需在 Network Observability 命名空间中手动复制其证书。原始证书会被监视，以便在需要时自动更新副本。([NETOBSERV-773](#))
- 在以前的版本中，Network Observability 代理没有涵盖 SCTP、ICMPv4 和 ICMPv6 协议，从而导致较少的全面的网络流覆盖。现在，这些协议可以被识别以改进流覆盖。([NETOBSERV-934](#))

28.1.1.5. 已知问题

- 当 **FlowCollector** 中的 **processor.metrics.tls** 设置为 **PROVIDED** 时，**flowlogs-pipeline servicemonitor** 不会适应 TLS 方案。([NETOBSERV-1087](#))

28.1.2. Network Observability Operator 1.2.0

以下公告可用于 Network Observability Operator 1.2.0：

- [RHSA-2023:1817 Network Observability Operator 1.2.0](#)

28.1.2.1. 准备下一次更新

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。在 Network Observability Operator 的 1.2 发布前，唯一可用的频道为 **v1.0.x**。Network Observability Operator 的 1.2 发行版本引入了用于跟踪和接收更新的 **stable** 更新频道。您必须将频道从 **v1.0.x** 切换到 **stable**，以接收将来的 Operator 更新。**v1.0.x** 频道已弃用，计划在以后的发行版本中删除。

28.1.2.2. 新功能及功能增强

28.1.2.2.1. 流量流视图中的直方图

- 现在，您可以选择显示一段时间内流的直方图表。histogram 可让您视觉化流历史记录，而不会达到 Loki 查询的限制。如需更多信息，请参阅[使用直方图](#)。

28.1.2.2.2. 对话跟踪

- 现在，您可以通过 **Log Type** 查询流，它允许对同一对话一部分的网络流进行分组。如需更多信息，请参阅[使用对话](#)。

28.1.2.2.3. Network Observability 健康警报

- 现在，如果因为写入阶段出现错误，或者达到 Loki ingestion 速率限制，Network Observability Operator 会在 **flowlogs-pipeline** 丢弃流时自动创建自动警报。如需更多信息，请参阅[查看健康信息](#)。

28.1.2.3. 程序错误修复

- 在以前的版本中，在更改 FlowCollector spec 中的 **namespace** 值后，在前一个命名空间中运行的 eBPF Agent pod 没有被适当删除。现在，在上一个命名空间中运行的 pod 会被正确删除。[\(NETOBSERV-774\)](#)
- 在以前的版本中，在更改 FlowCollector spec (如 Loki 部分) 中的 **caCert.name** 值后，FlowLogs-Pipeline pod 和 Console 插件 pod 不会重启，因此它们不知道配置更改。现在，pod 被重启，因此它们会获得配置更改。[\(NETOBSERV-772\)](#)
- 在以前的版本中，在不同节点上运行的 pod 间的网络流有时没有正确识别为重复，因为它们由不同的网络接口捕获。这会导致控制台插件中显示过量的指标。现在，流会被正确识别为重复，控制台插件会显示准确的指标。[\(NETOBSERV-755\)](#)
- 控制台插件中的 "reporter" 选项用于根据源节点或目标节点的观察点过滤流。在以前的版本中，无论节点观察点是什么，这个选项都会混合流。这是因为在节点级别将网络流错误地报告为 Ingress 或 Egress。现在，网络流方向报告是正确的。源观察点的 "reporter" 选项过滤器，或目标观察点如预期。[\(NETOBSERV-696\)](#)
- 在以前的版本中，对于配置为直接将流作为 gRPC+protobuf 请求发送的代理，提交的有效负载可能太大，并由处理器的 GRPC 服务器拒绝。这会在有非常高负载的场景中发生，且只会发生在一些代理配置中。代理记录错误消息，例如：`grpc: received message larger than max`。因此，缺少有关这些流的信息丢失。现在，当大小超过阈值时，gRPC 有效负载被分成多个信息。因此，服务器可以维护连接状态。[\(NETOBSERV-617\)](#)

28.1.2.4. 已知问题

- 在 Network Observability Operator 的 1.2.0 发行版本中，使用 Loki Operator 5.6，Loki 证书转换会定期影响 **flowlogs-pipeline** pod，并导致丢弃流而不是写入 Loki 的流程。一段时间后，问题会自行修正，但它仍然会在 Loki 证书转换过程中导致临时流数据丢失。[\(NETOBSERV-980\)](#)

28.1.2.5. 主要的技术变化

- 在以前的版本中，您可以使用自定义命名空间安装 Network Observability Operator。此发行版本引入了 **转换 Webhook**，它更改了 **ClusterServiceVersion**。由于这个变化，所有可用的命名空间不再被列出。另外，要启用 Operator 指标集合，无法使用与其他 Operator 共享的命名空间（如 **openshift-operators** 命名空间）。现在，Operator 必须安装在 **openshift-netobserv-operator** 命名空间中。如果您之前使用自定义命名空间安装 Network Observability Operator，则无法自动升级到新的 Operator 版本。如果您之前使用自定义命名空间安装 Operator，您必须

删除已安装的 Operator 实例，并在 **openshift-netobserv-operator** 命名空间中重新安装 Operator。务必要注意，对于 **FlowCollector**、Loki、Kafka 和其他插件，仍可使用自定义命名空间（如常用的 **netobserv** 命名空间）。([NETOBSERV-907](#))([NETOBSERV-956](#))

28.1.3. Network Observability Operator 1.1.0

以下公告可用于 Network Observability Operator 1.1.0：

- [RHSA-2023:0786 Network Observability Operator 安全公告更新](#)

Network Observability Operator 现在稳定，发行频道已升级到 **v1.1.0**。

28.1.3.1. 程序错误修复

- 在以前的版本中，除非 Loki **authToken** 配置被设置为 **FORWARD** 模式，否则不再强制执行身份验证，允许任何可以在 OpenShift Container Platform 集群中连接到 OpenShift Container Platform 控制台的用户，在没有身份验证的情况下检索流。现在，无论 Loki **authToken** 模式如何，只有集群管理员才能检索流。([BZ#2169468](#))

28.2. 关于网络可观察性

红帽为集群管理员提供 Network Observability Operator 来观察 OpenShift Container Platform 集群的网络流量。Network Observability Operator 使用 eBPF 技术创建网络流。然后，OpenShift Container Platform 信息会增强网络流，并存储在 Loki 中。您可以在 OpenShift Container Platform 控制台中查看和分析所存储的 netflow 信息，以进一步洞察和故障排除。

28.2.1. Network Observability Operator 的依赖项

Network Observability Operator 需要以下 Operator：

- Loki：您必须安装 Loki。Loki 是用于存储所有收集的流的后端。建议您通过为安装 Red Hat Loki Operator 来为 Network Observability Operator 安装 Loki。

28.2.2. Network Observability Operator 的可选依赖项

- Grafana：您可以使用 Grafana Operator 安装 Grafana 以使用自定义仪表板和查询功能。红帽不支持 Grafana Operator。
- Kafka：它在 OpenShift Container Platform 集群中提供可扩展性、弹性和高可用性。建议您使用 AMQ Streams Operator 安装 Kafka 进行大规模部署。

28.2.3. Network Observability Operator

Network Observability Operator 提供 Flow Collector API 自定义资源定义。流收集器实例是在安装过程中创建的，并启用网络流集合的配置。Flow Collector 实例部署 pod 和服务，它们组成一个监控管道，然后收集网络流并将其与 Kubernetes 元数据一起增强，然后再存储在 Loki 中。eBPF 代理作为 **daemonset** 对象部署，会创建网络流。

28.2.4. OpenShift Container Platform 控制台集成

OpenShift Container Platform 控制台集成提供了概述、拓扑视图和流量流表。

28.2.4.1. 网络 Observability 指标

OpenShift Container Platform 控制台提供 **Overview** 选项卡，它显示集群上网络流量流的整体聚合指标。信息可以被节点、命名空间、所有者、pod 和服务显示。过滤器和显示选项可以进一步优化指标。

在 **Observe** → **Dashboards** 中，**Netobserv** 仪表板提供了 OpenShift Container Platform 集群中网络流的快速概述。您可以在以下类别中查看网络流量指标的距离：

- 每个源和目标命名空间的顶流率(1分钟比率)
- 每个源和目标节点发出的前字节率(1分钟比率)
- 每个源和目标节点接收的前字节率(1分钟比率)
- 每个源和目标工作负载发出的前字节率(1分钟率)
- 每个源和目标工作负载接收的字节率(1分钟比率)
- 每个源和目标工作负载发送的最大数据包率(1分钟率)
- 每个源和目标工作负载接收的顶级数据包率(1分钟比率)

您可以通过更改 **ignoreTags** 列表，将 **FlowCollector spec.processor.metrics** 配置为添加或删除指标。有关可用标签的更多信息，请参阅[流收集器 API 参考](#)

另外，在 **Observe** → **Dashboards** 中，**Netobserv/Health** 仪表板提供有关 Operator 健康状况的指标。

28.2.4.2. Network Observability 拓扑视图

OpenShift Container Platform 控制台提供 **Topology** 选项卡，显示网络流的图形表示和流量数量。拓扑视图代表 OpenShift Container Platform 组件之间的流量，作为网络图。您可以使用过滤器和显示选项重新定义图形。您可以访问节点、命名空间、所有者、pod 和服务的信息。

28.2.4.3. 流量流表

流量流表视图为原始流、非聚合过滤选项和可配置列提供视图。OpenShift Container Platform 控制台提供 **流量流** 标签页，显示网络流的数据和流量数量。

28.3. 安装 NETWORK OBSERVABILITY OPERATOR

安装 Loki 是使用 Network Observability Operator 的先决条件。建议您使用 Loki Operator 安装 Loki；在 Network Observability Operator 安装前，需要以下步骤。

Loki Operator 集成了一个网关，该网关通过 Loki 实现多租户和身份验证，以进行数据流存储。**LokiStack** 资源管理 Loki，它是一个可扩展、高度可用、多租户日志聚合系统和使用 OpenShift Container Platform 身份验证的 Web 代理。**LokiStack** 代理使用 OpenShift Container Platform 身份验证来强制实施多租户，并方便在 Loki 日志存储中保存和索引数据。



注意

Loki Operator 也可用于[使用 LokiStack 进行日志](#)。Network Observability Operator 需要与 Logging 分开的一个专用的 LokiStack。

28.3.1. 安装 Loki Operator

建议使用 Loki Operator 版本 5.7 安装 Loki，此版本提供了使用 **openshift-network** 10nant 配置模式创建 LokiStack 实例的功能。它还为 Network Observability 提供完全自动的、集群内身份验证和授权支持。

先决条件

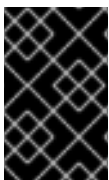
- 支持的日志存储 (AWS S3, Google Cloud Storage, Azure, Swift, Minio, OpenShift Data Foundation)
- OpenShift Container Platform 4.10+。
- Linux Kernel 4.18+。

您可以通过几种方法安装 Loki。其中一种方法是使用 OpenShift Container Platform Web 控制台 Operator Hub 安装 Loki Operator。

流程

1. 安装 **Loki Operator** 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 **Loki Operator**，然后点 **Install**。
 - c. 在 **Installation Mode** 下，选择 **All namespaces on the cluster**。
 - d. 验证您安装了 Loki Operator。访问 **Operators** → **Installed Operators** 页面，并查找 **Loki Operator**。
 - e. 验证 **Loki Operator** 是否在所有项目中 **Status** 为 **Succeeded**。
2. 创建 **Secret** YAML 文件。您可以通过 web 控制台或 CLI 中创建此 secret。
 - a. 使用 Web 控制台，进入到 **Project** → **All Projects** 下拉菜单，再选择 **Create Project**。将项目命名为 **netobserv**，再点 **Create**。
 - b. 使用右上角的 Import 图标+。将 YAML 文件放到编辑器中。在 **netobserv** 命名空间中创建此 YAML 文件非常重要，它使用 **access_key_id** 和 **access_key_secret** 指定您的凭证。
 - c. 创建 secret 后，您应该会在 web 控制台的 **Workloads** → **Secrets** 下看到它。
以下是一个 secret YAML 文件示例：

```
apiVersion: v1
kind: Secret
metadata:
  name: loki-s3
  namespace: netobserv
stringData:
  access_key_id: QUtJQUIPU0ZPRE5ON0VYQU1QTEUK
  access_key_secret:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRvhBTvBMRUtFWQo=
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```



重要

要卸载 Loki，请参考与用来安装 Loki 的方法相关的卸载过程。您可能会有剩余的 **ClusterRole** 和 **ClusterRoleBindings**、存储在对象存储中的数据，以及需要被删除的持久性卷。

28.3.1.1. 创建 LokiStack 自定义资源

建议您在由 FlowCollector 规格 `spec.namespace` 中引用的同一命名空间中部署 LokiStack。您可以使用 Web 控制台或 CLI 创建命名空间或新项目。

流程

1. 进入到 **Operators** → **Installed Operators**, 从 **Project** 下拉菜单查看 **All projects**。
2. 查找 **Loki Operator**。在详情的 **Provided APIs** 下, 选择 **LokiStack**。
3. 点 **Create LokiStack**。
4. 确保在 **Form View** 或 **YAML** 视图中指定以下字段：

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv
spec:
  size: 1x.small
  storage:
    schemas:
      - version: v12
        effectiveDate: '2022-06-01'
    secret:
      name: loki-s3
      type: s3
  storageClassName: gp3 1
  tenants:
    mode: openshift-network
```

- 1 使用集群中可用于 **ReadWriteOnce** 访问模式的存储类名称。您可以使用 **oc get storageclasses** 查看集群中的可用内容。



重要

您不能重复使用用于集群日志的相同的 LokiStack。

5. 点 **Create**。

28.3.1.1.1. 部署大小

Loki 的大小使用 **N<x>.<size>** 格式, 其中值 **<N>** 是实例的号, **<size>** 指定性能能力。



注意

1x.extra-small 仅用于演示目录, 它不被支持。

表 28.1. Loki 大小

	1x.extra-small	1x.small	1x.medium
数据传输	仅用于演示。	500GB/day	2TB/day
每秒查询数 (QPS)	仅用于演示。	25-50 QPS at 200ms	25-75 QPS at 200ms
复制因子	无	2	3
总 CPU 请求	5 个 vCPU	36 个 vCPU	54 个 vCPU
总内存请求	7.5Gi	63Gi	139Gi
总磁盘请求	150Gi	300Gi	450Gi

28.3.1.2. LokiStack ingestion 限制和健康警报

LokiStack 实例会根据配置的大小带有默认设置。您可以覆盖其中的一些设置，如 ingestion 和查询限制。如果您在 Console 插件中或 **flowlogs-pipeline** 日志中发现 Loki 错误，则可能需要更新它们。Web 控制台中的自动警报会在达到这些限制时通知您。

以下是配置的限制示例：

```
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 40
        ingestionRate: 20
        maxGlobalStreamsPerTenant: 25000
      queries:
        maxChunksPerQuery: 2000000
        maxEntriesLimitPerQuery: 10000
        maxQuerySeries: 3000
```

有关这些设置的更多信息，请参阅 [LokiStack API 参考](#)。

28.3.2. 配置授权和多租户

定义 **ClusterRole** 和 **ClusterRoleBinding**。**netobserv-reader ClusterRole** 启用多租户，并允许单独的用户访问或组访问 Loki 中的流。您可以创建一个 YAML 文件来定义这些角色。

流程

1. 使用 Web 控制台，点 Import 图标 +。
2. 将 YAML 文件放到编辑器中，然后点 **Create**：

ClusterRole reader yaml 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```

metadata:
  name: netobserv-reader ❶
rules:
- apiGroups:
  - 'loki.grafana.com'
  resources:
  - network
  resourceNames:
  - logs
  verbs:
  - 'get'

```

❶ 此角色可用于多租户。

ClusterRole writer yaml 示例

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: netobserv-writer
rules:
- apiGroups:
  - 'loki.grafana.com'
  resources:
  - network
  resourceNames:
  - logs
  verbs:
  - 'create'

```

ClusterRoleBinding yaml 示例

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: netobserv-writer-flp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: netobserv-writer
subjects:
- kind: ServiceAccount
  name: flowlogs-pipeline ❶
  namespace: netobserv
- kind: ServiceAccount
  name: flowlogs-pipeline-transformer
  namespace: netobserv

```

❶ **flowlogs-pipeline** 写入 Loki。如果使用 Kafka，这个值为 **flowlogs-pipeline-transformer**。

28.3.3. 在 Network Observability 中启用多租户

Network Observability Operator 中的多租户允许并将单独的用户访问或组访问限制为存储在 Loki 中的流。项目 admins 启用了访问权限。对某些命名空间具有有限访问权限的项目管理员只能访问这些命名空间的流。

前提条件

- 已安装 [Loki Operator 版本 5.7](#)
- **FlowCollector spec.loki.authToken** 配置必须设置为 **FORWARD**。
- 您必须以项目管理员身份登录

流程

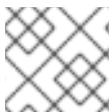
1. 运行以下命令，授权 **user1** 的读权限：

```
$ oc adm policy add-cluster-role-to-user netobserv-reader user1
```

现在，数据仅限于允许的用户命名空间。例如，可以访问单个命名空间的用户可以查看此命名空间内部的所有流，以及从到这个命名空间流进入这个命名空间。项目管理员有权访问 OpenShift Container Platform 控制台中的 Administrator 视角，以访问 Network Flows Traffic 页面。

28.3.4. 安装 Kafka（可选）

Kafka Operator 支持大规模环境。您可以从 Operator Hub 将 Kafka Operator 作为 [Red Hat AMQ Streams](#) 安装，就像 Loki Operator 和 Network Observability Operator 安装一样。



注意

要卸载 Kafka，请参考与用来安装的方法对应的卸载过程。

28.3.5. 安装 Network Observability Operator

您可以使用 OpenShift Container Platform Web 控制台 Operator Hub 安装 Network Observability Operator。安装 Operator 时，它提供 **FlowCollector** 自定义资源定义 (CRD)。在创建 **FlowCollector** 时，您可以在 web 控制台中设置规格。

先决条件

- 已安装 Loki。建议您使用 [Loki Operator 版本 5.7 安装 Loki](#)。
- 需要以下支持的架构之一：**amd64**, **ppc64le**, **arm64**, 或 **s390x**。
- Red Hat Enterprise Linux (RHEL) 9 支持的任何 CPU



注意

本文档假设您的 **LokiStack** 实例名称为 **loki**。使用其他名称需要额外的配置。

流程

1. 在 OpenShift Container Platform Web 控制台中，点击 **Operators** → **OperatorHub**。

2. 从 OperatorHub 中的可用 Operator 列表中选择 **Network Observability Operator**，然后点 **Install**。
3. 选中 **Enable Operator recommended cluster monitoring on this Namespace** 的复选框。
4. 导航到 **Operators → Installed Operators**。在 **Provided APIs for Network Observability** 下，选择 **Flow Collector** 链接。
 - a. 进入 **Flow Collector** 选项卡，然后点 **Create FlowCollector**。在表单视图中进行以下选择：
 - **spec.agent.ebpf.Sampling**：指定流的抽样大小。较低的抽样大小会对资源利用率造成负面影响。如需更多信息，请参阅 **spec.agent.ebpf** 下的 **FlowCollector** API 引用。
 - **spec.deploymentModel**：如果您使用 Kafka，请验证是否选择了 Kafka。
 - **spec.exporters**：如果您使用 Kafka，您可以选择将网络流发送到 Kafka，以便它们可以被支持 Kafka 输入的任何处理器或存储使用，如 Splunk、Elasticsearch 或 Fluentd。要做到这一点，请设置以下规格：
 - 将 **type** 设置为 **KAFKA**。
 - 将 **address** 设置为 **kafka-cluster-kafka-bootstrap.netobserv**。
 - 将 **topic** 设置为 **netobserv-flows-export**。Operator 将所有流导出到配置的 Kafka 主题。
 - 设置以下 **tls** 规格：
 - **certFile: service-ca.crt, name: kafka-gateway-ca-bundle**, 和 **type: configmap**。
您还可以通过直接编辑 YAML 来稍后配置这个选项。如需更多信息，请参阅 [导出增强的网络流数据](#)。
 - **loki.url**：由于身份验证被单独指定，此 URL 需要更新为 <https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network>。URL 的第一个部分"loki"应与 LokiStack 的名称匹配。
 - **loki.statusUrl**：将其设置为 <https://loki-query-frontend-http.netobserv.svc:3100/>。URL 的第一个部分"loki"应与 LokiStack 的名称匹配。
 - **loki.authToken**: 选择 **FORWARD** 值。
 - **tls.enable**：验证复选框是否已选中（代表已启用）。
 - **statusTls**：默认情况下，**enable** 值为 **false**。
对于证书引用名称的第一个部分：**loki-gateway-ca-bundle**、**loki-ca-bundle**、**loki-query-frontend-http**、**loki**、**loki** 的名称应与 **LokiStack** 的名称匹配。
 - b. 点 **Create**。

验证

要确认这一点，当您进入到 **Observe** 时，您应该看到选项中列出的 **Network Traffic**。

如果 OpenShift Container Platform 集群中没有应用程序流量，默认过滤器可能会显示"No results"，这会导致没有视觉流。在过滤器选择旁边，选择 **Clear all filters** 来查看流。



重要

如果使用 Loki Operator 安装 Loki，则建议不要使用 `querierUrl`，因为它可能会破坏对 Loki 的控制台访问。如果您安装的 Loki 是使用其他类型的 Loki 安装进行的，则不适用。

其他资源

- 有关流收集器规格的更多信息，请参阅[流收集器 API 参考](#)和[流收集器示例资源](#)。
- 有关将流数据导出到 Kafka 的更多信息，以了解第三方处理消耗，请参阅[导出增强的网络流数据](#)。

28.3.6. 卸载 Network Observability Operator

您可以使用 OpenShift Container Platform Web 控制台 Operator Hub 来卸载 Network Observability Operator，在 **Operators** → **Installed Operators** 区域中工作。

流程

1. 删除 **FlowCollector** 自定义资源。
 - a. 点 **Provided APIs** 列中的 **Network Observability Operator** 旁边的 **Flow Collector**。
 - b. 为**集群**点选项菜单 ，然后选择 **Delete FlowCollector**。
2. 卸载 Network Observability Operator。
 - a. 返回到 **Operators** → **Installed Operators** 区。
 - b. 点 **Network Observability Operator** 旁边的选项菜单  并选择 **Uninstall Operator**。
 - c. **Home** → **Projects** 并选择 **openshift-netobserv-operator**
 - d. 进入到 **Actions**，再选择 **Delete Project**
3. 删除 **FlowCollector** 自定义资源定义 (CRD)。
 - a. 进入到 **Administration** → **CustomResourceDefinitions**。
 - b. 查找 **FlowCollector** 并点选项菜单 。
 - c. 选择 **Delete CustomResourceDefinition**。



重要

如果已安装，Loki Operator 和 Kafka 会保留，需要被独立删除。另外，在一个对象存储中可能会有剩余的数据，以及持久性卷，这需要被删除。

28.4. OPENSIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR

Network Observability 是一个 OpenShift 操作器，它部署一个监控管道，以收集并增强网络流量流，由 Network Observability eBPF 代理生成。

28.4.1. 查看状态

Network Observability Operator 提供 Flow Collector API。创建 Flow Collector 资源时，它会部署 pod 和服务，以在 Loki 日志存储中创建和存储网络流，并在 OpenShift Container Platform Web 控制台中显示仪表盘、指标和流。

流程

1. 运行以下命令来查看 **Flowcollector** 的状态：

```
$ oc get flowcollector/cluster
```

输出示例

NAME	AGENT	SAMPLING (EBPF)	DEPLOYMENT MODEL	STATUS
cluster	EBPF	50	DIRECT	Ready

2. 输入以下命令检查在 **netobserv** 命名空间中运行的 pod 状态：

```
$ oc get pods -n netobserv
```

输出示例

NAME	READY	STATUS	RESTARTS	AGE
flowlogs-pipeline-56hbp	1/1	Running	0	147m
flowlogs-pipeline-9plvv	1/1	Running	0	147m
flowlogs-pipeline-h5gkb	1/1	Running	0	147m
flowlogs-pipeline-hh6kf	1/1	Running	0	147m
flowlogs-pipeline-w7vv5	1/1	Running	0	147m
netobserv-plugin-cdd7dc6c-j8ggp	1/1	Running	0	147m

flowlogs-pipeline pod 收集流，增强收集的流，然后将流发送到 Loki 存储。**netobserv-plugin** pod 为 OpenShift Container Platform 控制台创建一个视觉化插件。

1. 输入以下命令检查在 **netobserv-privileged** 命名空间中运行的 pod 状态：

```
$ oc get pods -n netobserv-privileged
```

输出示例

NAME	READY	STATUS	RESTARTS	AGE
netobserv-ebpf-agent-4lpp6	1/1	Running	0	151m
netobserv-ebpf-agent-6gbrk	1/1	Running	0	151m
netobserv-ebpf-agent-klp9	1/1	Running	0	151m
netobserv-ebpf-agent-vrcnf	1/1	Running	0	151m
netobserv-ebpf-agent-xf5jh	1/1	Running	0	151m

netobserv-ebpf-agent pod 监控节点的网络接口以获取流并将其发送到 **flowlogs-pipeline** pod。

1. 如果使用 Loki Operator，请输入以下命令检查在 **openshift-operators-redhat** 命名空间中运行的 pod 状态：

```
$ oc get pods -n openshift-operators-redhat
```

输出示例

```

NAME                                READY STATUS RESTARTS AGE
loki-operator-controller-manager-5f6cff4f9d-jq25h 2/2 Running 0      18h
lokistack-compactor-0                1/1 Running 0      18h
lokistack-distributor-654f87c5bc-qhkhv          1/1 Running 0      18h
lokistack-distributor-654f87c5bc-skxgm          1/1 Running 0      18h
lokistack-gateway-796dc6ff7-c54gz              2/2 Running 0      18h
lokistack-index-gateway-0                  1/1 Running 0      18h
lokistack-index-gateway-1                  1/1 Running 0      18h
lokistack-ingester-0                      1/1 Running 0      18h
lokistack-ingester-1                      1/1 Running 0      18h
lokistack-ingester-2                      1/1 Running 0      18h
lokistack-querier-66747dc666-6vh5x          1/1 Running 0      18h
lokistack-querier-66747dc666-cjr45          1/1 Running 0      18h
lokistack-querier-66747dc666-xh8rq          1/1 Running 0      18h
lokistack-query-frontend-85c6db4fbd-b2xfb      1/1 Running 0      18h
lokistack-query-frontend-85c6db4fbd-jm94f      1/1 Running 0      18h

```

28.4.2. 查看 Network Observability Operator 的状态和配置

您可以使用 **oc describe** 命令来检查 **FlowCollector** 的状态并查看其详情。

流程

1. 运行以下命令，以查看 Network Observability Operator 的状态和配置：

```
$ oc describe flowcollector/cluster
```

28.5. 配置 NETWORK OBSERVABILITY OPERATOR

您可以更新 Flow Collector API 资源，以配置 Network Observability Operator 及其受管组件。流收集器在安装过程中显式创建。由于此资源在集群范围内运行，因此只允许一个 **FlowCollector**，它必须被命名为 **cluster**。

28.5.1. 查看 FlowCollector 资源

您可以在 OpenShift Container Platform Web 控制台中直接查看和编辑 YAML。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。在这里，您可以修改 **FlowCollector** 资源来配置 Network Observability operator。

以下示例显示了 OpenShift Container Platform Network Observability operator 的 **FlowCollector** 资源示例：

抽样 FlowCollector 资源

```

apiVersion: flows.netobserv.io/v1beta1
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: DIRECT
  agent:
    type: EBPF 1
    ebpf:
      sampling: 50 2
      logLevel: info
      privileged: false
      resources:
        requests:
          memory: 50Mi
          cpu: 100m
        limits:
          memory: 800Mi
  processor:
    logLevel: info
    resources:
      requests:
        memory: 100Mi
        cpu: 100m
      limits:
        memory: 800Mi
    conversationEndTimeout: 10s
    logTypes: FLOWS 3
    conversationHeartbeatInterval: 30s
  loki: 4
    url: 'https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network'
    statusUrl: 'https://loki-query-frontend-http.netobserv.svc:3100/'
    authToken: FORWARD
    tls:
      enable: true
      caCert:
        type: configmap
        name: loki-gateway-ca-bundle
        certFile: service-ca.crt
  consolePlugin:
    register: true
    logLevel: info
    portNaming:
      enable: true
      portNames:
        "3100": loki
  quickFilters: 5
    - name: Applications
      filter:

```

```

src_namespace!: 'openshift-,netobserv'
dst_namespace!: 'openshift-,netobserv'
default: true
- name: Infrastructure
  filter:
    src_namespace: 'openshift-,netobserv'
    dst_namespace: 'openshift-,netobserv'
- name: Pods network
  filter:
    src_kind: 'Pod'
    dst_kind: 'Pod'
  default: true
- name: Services network
  filter:
    dst_kind: 'Service'

```

- 1 Agent 规格 **spec.agent.type** 必须是 **EBPF**。eBPF 是唯一的 OpenShift Container Platform 支持的选项。
- 2 您可以设置 Sampling 规格 **spec.agent.ebpf.sampling**，以管理资源。低抽样值可能会消耗大量计算、内存和存储资源。您可以通过指定一个抽样比率值来缓解这个问题。100 表示每 100 个流进行 1 个抽样。值 0 或 1 表示捕获所有流。数值越低，返回的流和派生指标的准确性会增加。默认情况下，eBPF 抽样设置为 50，因此每 50 个流抽样 1 个。请注意，更多抽样流也意味着需要更多存储。建议以默认值开始，并逐渐进行调整，以决定您的集群可以管理哪些设置。
- 3 可选规格 **spec.processor.logTypes**、**spec.processor.conversationHeartbeatInterval** 和 **spec.processor.conversationEndTimeout** 可以被设置为启用对话跟踪。启用后，可在 web 控制台中查询对话事件。**spec.processor.logTypes** 的值如下：**FLOWS**、**CONVERSATIONS**、**ENDED_CONVERSATIONS** 或 **ALL**。**ALL** 的存储要求最高，**ENDED_CONVERSATIONS** 的存储要求最低。
- 4 Loki 规格 **spec.loki** 指定 Loki 客户端。默认值与安装 Loki Operator 部分中提到的 Loki 安装路径匹配。如果您为 Loki 使用另一个安装方法，请为安装指定适当的客户端信息。
- 5 **spec.quickFilters** 规范定义了要在 web 控制台中显示的过滤器。**Application** 过滤器键 **src_namespace** 和 **dst_namespace** 是负的 (!)，因此 **Application** 过滤器显示不是来自、或目的地是 **openshift-** 或 **netobserv** 命名空间的所有流量。如需更多信息，请参阅配置快速过滤器。

其他资源

有关对话跟踪的更多信息，请参阅[使用对话](#)。

28.5.2. 使用 Kafka 配置流收集器资源

您可以将 **FlowCollector** 资源配置为使用 Kafka。需要运行一个 Kafka 实例，并在该实例中创建专用于 OpenShift Container Platform Network Observability 的 Kafka 主题。如需更多信息，请参阅您的 Kafka 文档，如 [AMQ Streams 的 Kafka 文档](#)。

以下示例演示了如何修改 OpenShift Container Platform Network Observability Operator 的 **FlowCollector** 资源以使用 Kafka：

FlowCollector 资源中的 Kafka 配置示例

```
deploymentModel: KAFKA
```

1

```
kafka:
  address: "kafka-cluster-kafka-bootstrap.netobserv" 2
  topic: network-flows 3
  tls:
    enable: false 4
```

- 1 将 `spec.deploymentModel` 设置为 **KAFKA** 而不是 **DIRECT** 来启用 Kafka 部署模型。
- 2 `spec.kafka.address` 是指 Kafka bootstrap 服务器地址。如果需要，您可以指定一个端口，如 `kafka-cluster-kafka-bootstrap.netobserv:9093`，以便在端口 9093 上使用 TLS。
- 3 `spec.kafka.topic` 应与 Kafka 中创建的主题名称匹配。
- 4 `spec.kafka.tls` 可用于加密所有与带有 TLS 或 mTLS 的 Kafka 的通信。启用后，Kafka CA 证书必须作为 ConfigMap 或 Secret 提供，两者都位于部署了 **flowlogs-pipeline** 处理器组件的命名空间中（默认为 **netobserv**）以及 eBPF 代理被部署的位置（默认为 **netobserv-privileged**）。它必须通过 `spec.kafka.tls.caCert` 引用。使用 mTLS 时，客户端 secret 还必须在这些命名空间中可用（它们也可以使用 AMQ Streams User Operator 生成并使用 `spec.kafka.tls.userCert`）。

28.5.3. 导出增强的网络流数据

您可以将网络流发送到 Kafka，以便它们可以被支持 Kafka 输入的任何处理器或存储使用，如 Splunk、Elasticsearch 或 Fluentd。

先决条件

- 安装的 Kafka

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 编辑 **FlowCollector** 以配置 `spec.exporters`，如下所示：

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  exporters:
    - type: KAFKA
      kafka:
        address: "kafka-cluster-kafka-bootstrap.netobserv"
        topic: netobserv-flows-export 1
        tls:
          enable: false 2
```

- 1 Network Observability Operator 将所有流导出到配置的 Kafka 主题。

- 2 您可以加密所有使用 SSL/TLS 或 mTLS 的 Kafka 的通信。启用后，Kafka CA 证书必须作为 ConfigMap 或 Secret 提供，两者都位于部署了 **flowlogs-pipeline** 处理器组件的命名空间中（默认为 netobserv）。它必须使用 **spec.exporters.tls.caCert** 引用。使用 mTLS 时，客户端 secret 还必须在这些命名空间中可用（它们也可以使用 AMQ Streams User Operator 生成并使用 **spec.exporters.tls.userCert** 引用）。

5. 配置后，网络流数据可以以 JSON 格式发送到可用输出。如需更多信息，请参阅 [网络流格式参考](#)

其他资源

有关指定流格式的更多信息，请参阅[网络流格式参考](#)。

28.5.4. 更新流收集器资源

作为在 OpenShift Container Platform Web 控制台中编辑 YAML 的替代方法，您可以通过修补 **flowcollector** 自定义资源 (CR) 来配置规格，如 eBPF 抽样：

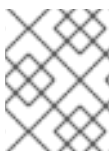
流程

1. 运行以下命令来修补 **flowcollector** CR 并更新 **spec.agent.ebpf.sampling** 值：

```
$ oc patch flowcollector cluster --type=json -p [{"op": "replace", "path":
"/spec/agent/ebpf/sampling", "value": <new value>}] -n netobserv"
```

28.5.5. 配置快速过滤器

您可以修改 **FlowCollector** 资源中的过滤器。可以使用双引号对值进行完全匹配。否则，会对文本值进行部分匹配。感叹号(!)字符放置在键的末尾，表示负效果。有关修改 YAML 的更多信息，请参阅示例 **FlowCollector** 资源。



注意

匹配类型 "all" 或 "any of" 的过滤器是用户可从查询选项进行修改的 UI 设置。它不是此资源配置的一部分。

以下是所有可用过滤器键的列表：

表 28.2. 过滤键

Universe*	源	目的地	描述
namespace	src_namespace	dst_namespace	过滤与特定命名空间相关的流量。
name	src_name	dst_name	过滤与给定叶资源名称相关的流量，如特定 pod、服务或节点（用于 host-network 流量）。
kind	src_kind	dst_kind	过滤与给定资源类型相关的流量。资源类型包括叶资源 (Pod、Service 或 Node) 或所有者资源 (Deployment 和 StatefulSet)。

Universal*	源	目的地	描述
owner_name	src_owner_name	dst_owner_name	过滤与给定资源所有者相关的流量；即工作负载或一组 pod。例如，它可以是 Deployment 名称、StatefulSet 名称等。
resource	src_resource	dst_resource	过滤与特定资源相关的流量，它们通过其规范名称表示，以唯一标识它。规范表示法是 kind.namespace.name 用于命名空间类型，或 node.name 用于节点。例如， Deployment.my-namespace.my-web-server 。
address	src_address	dst_address	过滤与 IP 地址相关的流量。支持 IPv4 和 IPv6。还支持 CIDR 范围。
mac	src_mac	dst_mac	过滤与 MAC 地址相关的流量。
端口	src_port	dst_port	过滤与特定端口相关的流量。
host_address	src_host_address	dst_host_address	过滤与运行 pod 的主机 IP 地址相关的流量。
protocol	不适用	不适用	过滤与协议相关的流量，如 TCP 或 UDP。

- 任何源或目的地的通用密钥过滤器。例如，过滤 **name: 'my-pod'** 表示从 **my-pod** 和所有流量到 **my-pod** 的所有流量，无论使用的匹配类型是什么，无论是 **匹配所有** 还是 **匹配任何**。

28.5.6. 资源管理和性能注意事项

Network Observability 所需的资源量取决于集群的大小以及集群要存储可观察数据的要求。要管理集群的资源并设置性能标准，请考虑配置以下设置。配置这些设置可能会满足您的最佳设置和可观察性需求。

以下设置可帮助您管理 outset 中的资源和性能：

eBPF Sampling

您可以设置 Sampling 规格 **spec.agent.ebpf.sampling**，以管理资源。较小的抽样值可能会消耗大量计算、内存和存储资源。您可以通过指定一个抽样比率值来缓解这个问题。**100** 表示每 100 个流进行 1 个抽样。值 **0** 或 **1** 表示捕获所有流。较小的值会导致返回的流和派生指标的准确性增加。默认情况下，eBPF 抽样设置为 50，因此每 50 个流抽样 1 个。请注意，更多抽样流也意味着需要更多存储。考虑以默认值开始，并逐步优化，以确定您的集群可以管理哪些设置。

限制或排除接口

通过为 **spec.agent.ebpf.interfaces** 和 **spec.agent.ebpf.excludeInterfaces** 设置值来减少观察到的流量。默认情况下，代理获取系统中的所有接口，但 **excludeInterfaces** 和 **lo**（本地接口）中列出的接口除外。请注意，接口名称可能会因使用的 Container Network Interface (CNI) 而异。

以下设置可用于在 Network Observability 运行后对性能进行微调：

资源要求和限制

使用 `spec.agent.ebpf.resources` 和 `spec.processor.resources` 规格，将资源要求和限制调整为集群中预期的负载和内存用量。800MB 的默认限制可能足以满足大多数中型集群。

缓存最大流超时

使用 eBPF 代理的 `spec.agent.ebpf.cacheMaxFlows` 和 `spec.agent.ebpf.cacheActiveTimeout` 规格来控制代理报告的频率。较大的值会导致代理生成较少的流量，与较低 CPU 负载相关联。但是，较大的值会导致内存消耗稍有更高的，并可能会在流集合中生成更多延迟。

28.5.6.1. 资源注意事项

下表概述了具有特定工作负载的集群的资源注意事项示例。



重要

表中概述的示例演示了为特定工作负载量身定制的场景。每个示例仅作为基准，可以进行调整以适应您的工作负载需求。

表 28.3. 资源建议

	Extra small (10 个节点)	Small (25 个节点)	Medium (65 个节点) [2]	Large (120 个节点) [2]
Worker 节点 vCPU 和内存	4 个 vCPU 16GiB 内存 [1]	16 个 vCPU 64GiB 内存 [1]	16 个 vCPU 64GiB 内存 [1]	16 个 vCPU 64GiB Mem [1]
LokiStack 大小	1x.extra-small	1x.small	1x.small	1x.medium
Network Observability 控制器内存限值	400Mi (默认)	400Mi (默认)	400Mi (默认)	800Mi
eBPF 抽样率	50 (默认)	50 (默认)	50 (默认)	50 (默认)
eBPF 内存限值	800Mi (默认)	800Mi (默认)	2000Mi	800Mi (默认)
FLP 内存限制	800Mi (默认)	800Mi (默认)	800Mi (默认)	800Mi (默认)
FLP Kafka 分区	不适用	48	48	48
Kafka 消费者副本	不适用	24	24	24
Kafka 代理	不适用	3 (默认)	3 (默认)	3 (默认)

1. 使用 AWS M6i 实例测试。
2. 除了此 worker 及其控制器外，还会测试 3 infra 节点 (size **M6i.12xlarge**) 和 1 个工作负载节点 (size **M6i.8xlarge**)。

28.6. NETWORK POLICY

作为具有 **admin** 角色的用户，您可以为 **netobserv** 命名空间创建网络策略。

28.6.1. 为 Network Observability 创建网络策略

您可能需要创建一个网络策略来保护到 **netobserv** 命名空间的入口流量。在 Web 控制台中，您可以使用表单视图创建网络策略。

流程

1. 进入到 **Networking → NetworkPolicies**。
2. 从 **Project** 下拉菜单中选择 **netobserv** 项目。
3. 为策略命名。在本例中，策略名为 **allow-ingress**。
4. 点 **Add ingress rule** 三次来创建三个入站规则。
5. 指定以下内容：
 - a. 对第一个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - b. 为第二个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - ii. 点 **+ Add namespace selector**。
 - iii. 添加标签 **kubernetes.io/metadata.name**，以及选择器 **openshift-console**。
 - c. 为第三个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - ii. 点 **+ Add namespace selector**。
 - iii. 添加标签 **kubernetes.io/metadata.name**，和选择器 **openshift-monitoring**。

验证

1. 进入到 **Observe → Network Traffic**。
2. 查看 **Traffic Flows** 选项卡，或任何标签页，以验证是否显示数据。
3. 进入到 **Observe → Dashboards**。在 **NetObserv/Health** 选择中，验证流是否嵌套并发送到 **Loki**，这在第一个图形中表示。

28.6.2. 网络策略示例

以下注解了 **netobserv** 命名空间的 **NetworkPolicy** 对象示例：

网络策略示例

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-ingress
  namespace: netobserv
spec:
  podSelector: {} ①
  ingress:
    - from:
      - podSelector: {} ②
        namespaceSelector: ③
          matchLabels:
            kubernetes.io/metadata.name: openshift-console
      - podSelector: {}
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-monitoring
  policyTypes:
    - Ingress
status: {}

```

- ① 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。在本文档中，它是安装 Network Observability Operator 的项目，即 **netobserv** 项目。
- ② 与策略对象允许从中入口流量的 pod 匹配的选择器。默认为选择器与 **NetworkPolicy** 相同的命名空间中的 pod 匹配。
- ③ 当指定 **namespaceSelector** 时，选择器与指定命名空间中的 pod 匹配。

其他资源

[使用 CLI 创建网络策略](#)

28.7. 观察网络流量

作为管理员，您可以观察 OpenShift Container Platform 控制台中的网络流量，以了解故障排除和分析的详细故障排除和分析。此功能帮助您从不同的流量流的图形表示获得见解。观察网络流量有几种可用的视图。

28.7.1. 从 Overview 视图观察网络流量

Overview 视图显示集群中网络流量流的整体聚合指标。作为管理员，您可以使用可用的显示选项监控统计信息。

28.7.1.1. 使用 Overview 视图

作为管理员，您可以进入到 **Overview** 视图来查看流速率统计的图形表示。

流程

1. 进入到 **Observe** → **Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **Overview** 选项卡。

您可以通过点菜单图标来配置每个流速率数据的范围。

28.7.1.2. 为 Overview 视图配置高级选项

您可以使用高级选项自定义图形视图。要访问高级选项，请点 **Show advanced options**。您可以使用 **Display options** 下拉菜单配置图形中的详情。可用的选项有：

- **Metric type**：在 **Bytes** 或 **Packets** 中显示的指标。默认值为 **Bytes**。
- **Scope**：选择网络流量流之间的组件详情。您可以将范围设置为 **Node**、**Namespace**、**Owner** 或 **Resource**。**Owner** 是一个资源聚合。**Resource** 可以是一个 pod、服务、节点（主机网络流量），或未知 IP 地址。默认值为 **Namespace**。
- **Truncate labels**：从下拉列表中选择标签所需的宽度。默认值为 **M**。

28.7.1.2.1. 管理面板

您可以选择显示所需的统计信息，并对它们进行重新排序。若要管理列，点 **Manage panels**。

28.7.2. 从流量流视图观察网络流量

流量流 视图显示网络流的数据以及表中的流量数量。作为管理员，您可以使用流量流表监控应用程序间的流量数量。

28.7.2.1. 使用流量流视图

作为管理员，您可以进入 **流量流** 表来查看网络流信息。

流程

1. 进入到 **Observe** → **Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **流量流** 选项卡。

您可以点击每行来获取对应的流信息。

28.7.2.2. 为流量流视图配置高级选项

您可以使用 **Show advanced options** 自定义和导出视图。您可以使用 **Display options** 下拉菜单设置行大小。默认值为 **Normal**。

28.7.2.2.1. 管理列

您可以选择显示所需的列，并对它们进行重新排序。若要管理列，可点 **Manage 列**。

28.7.2.2.2. 导出流量流数据

您可以从**流量流**视图导出数据。

流程

1. 点 **Export data**。

2. 在弹出窗口中，您可以选择 **Export all data** 复选框，以导出所有数据，然后清除复选框以选择要导出的必填字段。
3. 单击 **Export**。

28.7.2.3. 使用对话跟踪

作为管理员，您可以对属于同一对话的网络流进行分组。对话被定义为一组由 IP 地址、端口和协议标识的对等点，从而产生唯一的 **Conversation Id**。您可以在 web 控制台中查询对话事件。这些事件在 web 控制台中表示，如下所示：

- **Conversation start**：连接启动或 TCP 标志被截获时发生此事件
- **Conversation tick**：此事件在连接处于活跃状态时根据 **FlowCollector spec.processor.conversationHeartbeatInterval** 参数中定义每个指定间隔发生。
- **Conversation end**：当达到 **FlowCollector spec.processor.conversationEndTimeout** 参数或 TCP 标志被截获时，会发生此事件。
- **Flow**：这是在指定间隔内的网络流量流。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 配置 **FlowCollector** 自定义资源，以便根据您的观察需求设置 **spec.processor.logTypes**，**conversationEndTimeout**，和 **conversationHeartbeatInterval** 参数。示例配置示例如下：

配置 FlowCollector 以对话跟踪

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  processor:
    conversationEndTimeout: 10s
    logTypes: FLOWS
    conversationHeartbeatInterval: 30s
```

- 1 **Conversation end** 事件表示达到了 **conversationEndTimeout**，或 TCP 标志被截获。
- 2 当 **logTypes** 设置为 **FLOWS** 时，只导出 **Flow** 事件。如果将值设为 **ALL**，则检索和流事件都会在 **Network Traffic** 页面中导出并可见。要只专注于对话事件，您可以指定 **CONVERSATIONS**，这会导出 **Conversation start**，**Conversation tick** 和 **Conversation end** 时间；或 **ENDED_CONVERSATIONS**，这会只导出 **Conversation end**。**ALL** 的存储要求最高，**ENDED_CONVERSATIONS** 的存储要求最低。
- 3 **Conversation tick** 事件表示当网络连接活跃时，在 **FlowCollector conversationHeartbeatInterval** 参数中定义的每个指定间隔。



注意

如果您更新了 **logType** 选项，则之前选择中的流不会从控制台插件中清除。例如，如果您最初将 **logType** 设置为 **CONVERSATIONS** 来将时间分散直到 10 AM，然后移到 **ENDED_CONVERSATIONS**，控制台插件会显示 10M 之前的所有对话事件，且仅在 10M 后终止对话。

5. 刷新 **Traffic flows** 标签页中的 **Network Traffic**。通知请注意，有两个新列：**Event/Type** 和 **Conversation Id**。当 **Flow** 是所选查询选项时，所有 **Event/Type** 字段都是 **Flow**。
6. 选择 **Query Options** 并选择 **Log Type,Conversation**。现在，**Event/Type** 会显示所有所需的对话事件。
7. 接下来，您可以过滤侧面板中的 **Conversation** 和 **Flow** 日志类型选项的特定对话 ID 或切换。

28.7.2.3.1. 使用直方图

您可以点 **Show histogram** 来显示工具栏视图，以使用栏图的形式可视化流历史记录。histogram 显示一段时间内的日志数量。您可以选择直方图的一部分在下面的工具栏中过滤网络流数据。

28.7.3. 从 Topology 视图中观察网络流量

Topology 视图提供了网络流和流量数量的图形表示。作为管理员，您可以使用 **Topology** 视图监控应用程序间的流量数据。

28.7.3.1. 使用 Topology 视图

作为管理员，您可以进入到 **Topology** 视图来查看组件的详情和指标。

流程

1. 进入到 **Observe** → **Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **Topology** 选项卡。

您可以点 **Topology** 中的每个组件来查看组件的详情和指标。

28.7.3.2. 为 Topology 视图配置高级选项

您可以使用 **Show advanced options** 自定义和导出视图。高级选项视图具有以下功能：

- **Find in view**: 要在视图中搜索所需组件。
- **Display options** : 要配置以下选项：
 - **Layout** : 要选择图形表示的布局。默认值为 **ColaNoForce**。
 - **Scope** : 选择网络流量流之间的组件范围。默认值为 **Namespace**。
 - **组** : 通过对组件进行分组来充分了解所有权。默认值为 **None**。
 - **折叠组** : 要展开或折叠组。默认会扩展组。如果 **组** 的值为 **None**，这个选项会被禁用。
 - **显示** : 要选择需要显示的详细信息。默认检查所有选项。可以选项为：**Edges, Edges label**, 和 **Badges**。

- **Truncate labels** : 从下拉列表中选择标签所需的宽度。默认值为 **M**。

28.7.3.2.1. 导出拓扑视图

要导出视图，点 **Export topology view**。该视图以 PNG 格式下载。

28.7.4. 过滤网络流量

默认情况下，Network Traffic 页面根据 **FlowCollector** 实例中配置的默认过滤器显示集群中的流量流数据。您可以通过更改 preset 过滤器，使用过滤器选项观察所需的数据。

查询选项

您可以使用 **Query Options** 来优化搜索结果，如下所示：

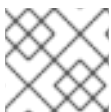
- **日志类型** : 可用选项 **Conversation** 和 **Flows** 提供了按日志类型查询流的能力，如流日志、新对话、完成对话和心跳，这是长期对话的定期记录。对话是同一对等点之间的流聚合。
- **Reporter Node** : 可以从源和目标节点报告每个流。对于集群入口，从目标节点和集群出口报告流，从源节点报告流。您可以选择 **Source** 或 **Destination**。**Overview** 和 **Topology** 视图中禁用了 **Both** 选项。默认选择的值是 **Destination**。
- **Match filters** : 您可以确定高级过滤器中选择的过滤器参数之间的关系。可用的选项包括 **Match all** 和 **Match any**。**Match all** 提供与所有值都匹配的结果，而 **Match any** 则提供与输入的任何值匹配的结果。默认值为 **Match all**。
- **Limit** : 内部后端查询的数据限制。根据匹配和过滤器设置，流量流数据的数量显示在指定的限制中。

快速过滤器

Quick 过滤器 下拉菜单中的默认值在 **FlowCollector** 配置中定义。您可从控制台修改选项。

高级过滤器

您可以通过提供要过滤的参数及其相应的文本值来设置高级过滤器。参数下拉列表中的 **Common** 部分会过滤与 **Source** 或 **Destination** 匹配的结果。要启用或禁用应用的过滤器，您可以点过滤器选项下面列出的应用过滤器。



注意

要了解指定文本值的规则，请点[了解更多](#)。

您可以点 **Reset default filter** 删除现有过滤器，并应用 **FlowCollector** 配置中定义的过滤器。

另外，您也可以访问 **Namespaces, Services, Routes, Nodes, and Workloads** 页中的 **Network Traffic** 标签页，它们提供了相关部分的聚合过滤数据。

其他资源

有关在 **FlowCollector** 中配置快速过滤器的更多信息，请参阅[配置快速过滤器](#) 和 [流收集器示例资源](#)。

28.8. 监控 NETWORK OBSERVABILITY OPERATOR

您可以使用 Web 控制台监控与 Network Observability Operator 健康相关的警报。

28.8.1. 查看健康信息

您可从 web 控制台的 **Dashboards** 页面中访问 Network Observability Operator 健康和资源使用情况的指标。当触发警报时，您定向到仪表板的健康警报横幅可能会出现在 **Network Traffic** 和 **Home** 页面中。在以下情况下生成警报：

- 如果 **flowlogs-pipeline** 工作负载因为 Loki 错误而丢弃流，如已经达到 Loki ingestion 速率限制，则 **NetObservLokiError** 警报发生。
- 如果没有在一定时间内计算流，则 **NetObservNoFlows** 警报发生。先决条件
- 已安装 Network Observability Operator。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群，或者具有所有项目的查看权限。

流程

1. 从 web 控制台中的 **Administrator** 视角，进入到 **Observe → Dashboards**。
2. 从 **Dashboards** 下拉菜单中选择 **Netobserv/Health**。有关 Operator 健康的指标显示在页面中。

28.8.1.1. 禁用健康警报

您可以通过编辑 **FlowCollector** 资源来选择不使用健康警报：

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 添加 **spec.processor.metrics.disableAlerts** 来禁用健康警报，如下例所示：

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  processor:
    metrics:
      disableAlerts: [NetObservLokiError, NetObservNoFlows] ❶
```

- ❶ 您可以指定一个或多个包含要禁用的警报类型的列表。

28.9. FLOWCOLLECTOR 配置参数

FlowCollector 是网络流集合 API 的 Schema，它试用并配置底层部署。

28.9.1. FlowCollector API 规格

描述

FlowCollector 是网络流集合 API 的 schema，它试用并配置底层部署。

类型

对象

属性	类型	描述
apiVersion	字符串	APIVersion 定义对象的这个表示法的版本化的 schema。服务器应该将识别的模式转换为最新的内部值，并可拒绝未识别的值。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
kind	字符串	kind 是一个字符串值，代表此对象所代表的 REST 资源。服务器可以从客户端向其提交请求的端点推断。无法更新。采用驼峰拼写法 (CamelCase)。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds
metadata	object	标准对象元数据。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
spec	object	FlowCollectorSpec 定义 FlowCollector 资源的所需状态。 *：本文中“不支持”或“弃用”的功能意味着，红帽不正式支持此功能。例如，它可能已被社区贡献并接受，而无需正式协议进行维护。产品维护人员可能只为这些功能提供一些支持。

28.9.1.1. .metadata

描述

标准对象元数据。更多信息：<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

类型

object

28.9.1.2. .spec

描述

FlowCollectorSpec 定义 FlowCollector 资源的所需状态。

* : 本文档中 "不支持"或"弃用" 的功能意味着, 红帽不正式支持此功能。例如, 它可能已被社区贡献并接受, 而无需正式协议进行维护。产品维护人员可能只为这些功能提供一些支持。

类型

对象

必需

- **agent**
- **deploymentModel**

属性	类型	描述
agent	object	流提取的代理配置。
consolePlugin	object	ConsolePlugin 定义与 OpenShift Container Platform 控制台插件相关的设置 (如果可用)。
deploymentModel	字符串	deploymentModel 定义所需的用于流处理的部署类型。可能的值有 : - DIRECT (默认) 使流处理器直接从代理侦听。 - KAFKA - KAFKA 在处理器消耗前发送到 Kafka 管道。 Kafka 可以提供更好的可扩展性、弹性和高可用性 (更多详情, 请参阅 https://www.redhat.com/en/topics/integration/what-is-apache-kafka)。
exporters	数组	exporters 为自定义消耗或存储定义了额外的可选导出器。
kafka	object	Kafka 配置, 允许使用 Kafka 作为流集合管道的一部分。当 spec.deploymentModel 为 KAFKA 时可用。
loki	object	Loki, 流存储、客户端设置。
namespace	字符串	部署 NetObserv pod 的命名空间。如果为空, 则将使用 Operator 的命名空间。

属性	类型	描述
processor	object	processor 定义从代理接收流的组件设置，增强它们，生成指标，并将它们转发到 Loki 持久层和/或任何可用的导出器。

28.9.1.3. .spec.agent

描述

流提取的代理配置。

类型

对象

必填

- **type**

属性	类型	描述
ebpf	object	ebpf 描述了当 spec.agent.type 设置为 EBPF 时，与基于 eBPF 的流报告程序相关的设置。
ipfix	object	ipfix - 已弃用(*) - 描述了当 spec.agent.type 设置为 IPFIX 时，与基于 IPFIX 的流报告程序相关的设置。
type	字符串	type 选择流追踪代理。可能的值有： - EBPF （默认）使用 NetObserv eBPF 代理。 - IPFIX - 已弃用(*) - 使用旧的 IPFIX 收集器。 建议使用 EBPF ，因为它提供更好的性能，并应该可以正常工作，无论集群中是否安装了 CNI。 IPFIX 与 OVN-Kubernetes CNI 一起工作（如果支持导出 IPFIX，则其他 CNI 可以正常工作，但需要手动配置）。

28.9.1.4. .spec.agent.ebpf

描述

ebpf 描述了当 **spec.agent.type** 设置为 **EBPF** 时，与基于 eBPF 的流报告程序相关的设置。

类型

对象

属性	类型	描述
cacheActiveTimeout	字符串	cacheActiveTimeout 是报告者在发送前聚合流的最大周期。增加 cacheMaxFlows 和 cacheActiveTimeout 可能会降低网络流量开销和 CPU 负载，但您可以预期更高的内存消耗和流集合中的延迟增加。
cacheMaxFlows	整数	cacheMaxFlows 是聚合中的最大流数；达到时，报告者会发送流。增加 cacheMaxFlows 和 cacheActiveTimeout 可能会降低网络流量开销和 CPU 负载，但您可以预期更高的内存消耗和流集合中的延迟增加。
debug	object	debug 允许设置 eBPF 代理的内部配置的一些方面。本节专门用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS env vars。用户设置其值会自行承担任何风险。
excludeInterfaces	数组（字符串）	excludeInterfaces 包含从流追踪中排除的接口名称。条目用斜杠括起，如 /br- ，与正则表达式匹配。否则，它将匹配为区分大小写的字符串。
imagePullPolicy	字符串	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略
interfaces	数组（字符串）	interfaces 包含从中收集流的接口名称。如果为空，代理将获取系统中的所有接口，但 ExcludeInterfaces 中列出的接口除外。条目用斜杠括起，如 /br- ，与正则表达式匹配。否则，它将匹配为区分大小写的字符串。
kafkaBatchSize	整数	kafkaBatchSize 在发送到分区前限制请求的最大大小（以字节为单位）。如果不使用 Kafka，则忽略。默认：10MB。
logLevel	字符串	logLevel 定义 NetObserv eBPF 代理的日志级别

属性	类型	描述
privileged	布尔值	eBPF Agent 容器的特权模式。通常，此设置可以忽略或设置为 false：在这种情况下，Operator 会将粒度功能(BPF、PERFMON、NET_ADMIN、SYS_RESOURCE) 设置为容器，以启用其正确的操作。如果出于某种原因而无法设置这些功能，例如，如果旧的内核版本不知道 CAP_BPF，那么您可以打开此模式以获取更多全局权限。
资源	object	resources 是此容器所需的计算资源。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
sampling	整数	流报告器的抽样率。100 表示发送 100 个流中的一个。0 或 1 表示所有流都是抽样的。

28.9.1.5. .spec.agent.ebpf.debug

描述

debug 允许设置 eBPF 代理的内部配置的一些方面。本节专门用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS env vars。用户设置其值会自行承担任何风险。

类型

对象

属性	类型	描述
env	对象（字符串）	env 允许将自定义环境变量传递给底层组件。对于传递一些非常严格的性能调整选项（如 GOGC、GOMAXPROCS）非常有用，它们不应作为 FlowCollector 描述符的一部分公开，因为它们仅在边缘调试或支持场景中有用。

28.9.1.6. .spec.agent.ebpf.resources

描述

resources 是此容器所需的计算资源。更多信息：
<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型

对象

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manager-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。更多信息： https://kubernetes.io/docs/concepts/configuration/manager-resources-containers/

28.9.1.7. .spec.agent.ipfix

描述

ipfix - 已弃用 (*) - 描述了当 **spec.agent.type** 设置为 IPFIX 时，与基于 IPFIX 的流报告程序相关的设置。

类型

对象

属性	类型	描述
cacheActiveTimeout	字符串	cacheActiveTimeout 是报告者在发送前聚合流的最大周期
cacheMaxFlows	整数	cacheMaxFlows 是聚合中的最大流数；达到时，报告者会发送流
clusterNetworkOperator	object	clusterNetworkOperator 定义与 OpenShift Container Platform Cluster Network Operator 相关的设置（如果可用）。
forceSampleAll	布尔值	forceSampleAll 允许在基于 IPFIX 的流报告程序中禁用抽样。不建议通过 IPFIX 对所有流量进行抽样，因为它可能会生成集群不稳定。如果 REALLY 希望这样做，请将此标志设置为 true。自行使用风险。当设为 true 时， sampling 的值将被忽略。

属性	类型	描述
ovnKubernetes	object	ovnKubernetes 定义 OVN-Kubernetes CNI 的设置（如果可用）。在没有 OpenShift Container Platform 的情况下使用 OVN 的 IPFIX 导出时使用此配置。使用 OpenShift Container Platform 时，请参阅 clusterNetworkOperator 属性。
sampling	整数	sampling 是 reporter 的抽样率。100 表示发送 100 个流中的一个。为确保集群稳定性，它不能设置为低于 2 的值。如果您真正希望对影响集群稳定性的每个数据包进行抽样，请参阅 forceSampleAll 。或者，您可以使用 eBPF 代理而不是 IPFIX。

28.9.1.8. .spec.agent.ipfix.clusterNetworkOperator

描述

clusterNetworkOperator 定义与 OpenShift Container Platform Cluster Network Operator 相关的设置（如果可用）。

类型

对象

属性	类型	描述
namespace	字符串	部署配置映射的命名空间。

28.9.1.9. .spec.agent.ipfix.ovnKubernetes

描述

ovnKubernetes 定义 OVN-Kubernetes CNI 的设置（如果可用）。在没有 OpenShift Container Platform 的情况下使用 OVN 的 IPFIX 导出时使用此配置。使用 OpenShift Container Platform 时，请参阅 **clusterNetworkOperator** 属性。

类型

对象

属性	类型	描述
containerName	字符串	containerName 定义要为 IPFIX 配置的容器的名称。

属性	类型	描述
daemonSetName	字符串	daemonSetName 定义控制 OVN-Kubernetes pod 的 DaemonSet 的名称。
namespace	字符串	部署 OVN-Kubernetes pod 的命名空间。

28.9.1.10. .spec.consolePlugin

描述

ConsolePlugin 定义与 OpenShift Container Platform 控制台插件相关的设置（如果可用）。

类型

对象

属性	类型	描述
autoscaler	object	Pod 横向自动扩展的 autoscaler 规格来为插件部署设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2)。
imagePullPolicy	字符串	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略
logLevel	字符串	控制台插件后端的 logLevel
端口	整数	port 是插件服务端口。不要使用 9002，它为指标保留。
portNaming	object	portNaming 定义端口到服务名称转换的配置
quickFilters	数组	quickFilters 为 Console 插件配置快速过滤器预设置

属性	类型	描述
register	布尔值	register 允许（设置为 true 时），在 OpenShift Container Platform Console Operator 中自动注册提供的控制台插件。当设置为 false 时，您仍然可以使用以下命令编辑 console.operator.openshift.io/cluster 来手动注册它： oc patch console.operator.openshift.io cluster --type='json' -p '[{"op": "add", "path": "/spec/plugins/-", "value": "netobserv-plugin"}]'
replicas	整数	replicas 定义要启动的副本 (pod) 数。
资源	object	resources ，根据此容器所需的计算资源。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

28.9.1.11. .spec.consolePlugin.autoscaler

描述

Pod 横向自动扩展的 **autoscaler** 规格来为插件部署设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2)。

类型

object

28.9.1.12. .spec.consolePlugin.portNaming

描述

portNaming 定义端口到服务名称转换的配置

类型

对象

属性	类型	描述
enable	布尔值	启用 console 插件端口到服务名称转换
portNames	对象（字符串）	portNames 定义了在中使用的额外端口名称，例如 portNames: {"3100": "loki"} 。

28.9.1.13. .spec.consolePlugin.quickFilters

描述

quickFilters 为 Console 插件配置快速过滤器预设置

类型

数组

28.9.1.14. .spec.consolePlugin.quickFilters[]

描述

QuickFilter 为控制台的快速过滤器定义预设置配置

类型

对象

必需

- **filter**
- **name**

属性	类型	描述
default	布尔值	default 定义默认是否应激活此过滤器
filter	对象 (字符串)	filter 是一组在选择此过滤器时要设置的键和值。每个键都可以与使用组合字符串的值列表相关，例如 filter: {"src_namespace": "namespace1,namespace2"} 。
name	字符串	过滤器的名称，它将在控制台中显示

28.9.1.15. .spec.consolePlugin.resources

描述

resources，根据此容器所需的计算资源。更多信息：

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型

对象

属性	类型	描述
----	----	----

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

28.9.1.16. .spec.exporters

描述

exporters 为自定义消耗或存储定义了额外的可选导出器。

类型

数组

28.9.1.17. .spec.exporters[]

描述

FlowCollectorExporter 定义了一个额外的导出器来发送增强的流

类型

对象

必填

- **type**

属性	类型	描述
ipfix	object	IPFIX 配置，如 IP 地址和端口，以将增强的 IPFIX 流发送到。不支持 (*)。
kafka	object	Kafka 配置（如地址和主题）将增强的流发送到。
type	字符串	type 选择导出器类型。可用的选项有 KAFKA 和 IPFIX 。IPFIX 不被支持 (*)。

28.9.1.18. .spec.exporters[].ipfix

描述

IPFIX 配置，如 IP 地址和端口，以将增强的 IPFIX 流发送到。不支持(*)。

类型

对象

必需

- **targetHost**
- **targetPort**

属性	类型	描述
targetHost	字符串	IPFIX 外部接收器的地址
targetPort	整数	IPFIX 外部接收器的端口
传输	字符串	用于 IPFIX 连接的传输协议(TCP 或 UDP)，默认为 TCP 。

28.9.1.19. .spec.exporters[].kafka

描述

Kafka 配置（如地址和主题）将增强的流发送到。

类型

对象

必填

- **address**
- **topic**

属性	类型	描述
address	字符串	Kafka 服务器的地址
tls	object	TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。请注意，当使用 eBPF 代理时，Kafka 证书需要复制到代理命名空间中（默认为 netobserv-privileged ）。
topic	字符串	要使用的 Kafka 主题。它必须存在，NetObserv 将不会创建它。

28.9.1.20. .spec.exporters[.].kafka.tls

描述

TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。请注意，当使用 eBPF 代理时，Kafka 证书需要复制到代理命名空间中（默认为 **netobserv-privileged**）。

类型

对象

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

28.9.1.21. .spec.exporters[.].kafka.tls.caCert

描述

caCert 定义证书颁发机构的证书引用

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称

属性	类型	描述
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.22. .spec.exporters[].kafka.tls.userCert

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.23. .spec.kafka

描述

Kafka 配置，允许使用 Kafka 作为流集合管道的一部分。当 **spec.deploymentModel** 为 **KAFKA** 时可用。

类型

对象

必填

- **address**
- **topic**

属性	类型	描述
address	字符串	Kafka 服务器的地址
tls	object	TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。请注意，当使用 eBPF 代理时，Kafka 证书需要复制到代理命名空间中（默认为 netobserv-privileged ）。
topic	字符串	要使用的 Kafka 主题。它必须存在，NetObserv 将不会创建它。

28.9.1.24. .spec.kafka.tls

描述

TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。请注意，当使用 eBPF 代理时，Kafka 证书需要复制到代理命名空间中（默认为 **netobserv-privileged**）。

类型

对象

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

28.9.1.25. .spec.kafka.tls.caCert

描述

caCert 定义证书颁发机构的证书引用

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.26. .spec.kafka.tls.userCert

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称

属性	类型	描述
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.27. .spec.loki

描述

Loki，流存储、客户端设置。

类型

对象

属性	类型	描述
authToken	字符串	authToken 描述了获取向 Loki 进行身份验证的令牌。 - DISABLED 不会发送带有请求的任何令牌。 - FORWARD 将转发授权的用户令牌。 - HOST - 弃用(*) RTT - 将使用本地 pod 服务帐户向 Loki 进行身份验证。 使用 Loki Operator 时，必须设置为 FORWARD 。
batchSize	整数	batchSize 是发送前要积累的日志的最大批处理大小（以字节为单位）。
batchWait	字符串	batchWait 是发送批处理前等待的最长时间。
maxBackoff	字符串	maxBackoff 是重试之间客户端连接的最大 backoff 时间。
maxRetries	整数	maxRetries 是客户端连接的最大重试次数。
minBackoff	字符串	minBackoff 是重试之间客户端连接的初始 backoff 时间。

属性	类型	描述
querierUrl	字符串	querierUrl 指定 Loki querier 服务的地址，如果它与 Loki ingester URL 不同。如果为空，则使用 URL 值（假设 Loki ingester 和 querier 位于同一服务器中）。使用 Loki Operator 时，请不要设置它，因为 ingestion 和 query 使用 Loki 网关。
staticLabels	对象（字符串）	staticLabels 是每个流上设置的通用标签映射。
statusTls	object	Loki 状态 URL 的 TLS 客户端配置。
statusUrl	字符串	statusUrl 指定 Loki /ready 、 /metrics 和 /config 端点的地址，如果它与 Loki querier URL 不同。如果为空，则使用 querierURL 值。这可用于在前端中显示错误消息和一些上下文。使用 Loki Operator 时，将其设置为 Loki HTTP 查询前端服务，例如 https://loki-query-frontend-http.netobserv.svc:3100/ 。当设置 statusUrl 时，将使用 statusTLS 配置。
tenantID	字符串	tenantId 是 Loki X-Scope-OrgID ，用于标识每个请求的租户。使用 Loki Operator 时，将其设置为 network ，这对应于一个特殊的租户模式。
timeout	字符串	timeout 是最大时间连接/请求限制。超时值为零表示没有超时。
tls	object	Loki URL 的 TLS 客户端配置。
url	字符串	url 是要推送流至的现有 Loki 服务的地址。使用 Loki Operator 时，使用路径中设置的 network 租户将其设置为 Loki 网关服务，例如 https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network 。

28.9.1.28. .spec.loki.statusTls

描述

Loki 状态 URL 的 TLS 客户端配置。

类型

对象

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

28.9.1.29. .spec.loki.statusTls.caCert

描述

caCert 定义证书颁发机构的证书引用

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.30. .spec.loki.statusTls.userCert

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.31. .spec.loki.tls

描述

Loki URL 的 TLS 客户端配置。

类型

对象

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true，则忽略 caCert 字段。

属性	类型	描述
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

28.9.1.32. .spec.loki.tls.caCert

描述

caCert 定义证书颁发机构的证书引用

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.33. .spec.loki.tls.userCert

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

对象

属性	类型	描述
----	----	----

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.34. .spec.processor

描述

processor 定义从代理接收流的组件设置，增强它们，生成指标，并将它们转发到 Loki 持久层和/或任何可用的导出器。

类型

对象

属性	类型	描述
conversationEndTimeout	字符串	conversationEndTimeout 是收到网络流后等待的时间，请考虑对话结束的时间。为 TCP 流收集 FIN 数据包时会忽略此延迟（请参阅 conversationTerminatingTimeout ）。
conversationHeartbeatInterval	字符串	conversationHeartbeatInterval 是对话的 "tick" 事件间的等待时间

属性	类型	描述
conversationTerminatingTimeout	字符串	conversationTerminatingTimeout 是从检测到的 FIN 标志到对话结束间的等待时间。只适用于 TCP 流。
debug	object	debug 允许设置流处理器的内部配置。本节专门用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS env vars。用户设置其值会自行承担任何风险。
dropUnusedFields	布尔值	dropUnusedFields 允许，当设为 true 时，可以丢弃由 OVS 未使用的字段，以节省存储空间。
enableKubeProbes	布尔值	enableKubeProbes 是一个用于启用或禁用 Kubernetes 存活度和就绪度探测的标志
healthPort	整数	healthPort 是 Pod 中的收集器 HTTP 端口，用于公开健康检查 API
imagePullPolicy	字符串	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略
kafkaConsumerAutoscaler	object	kafkaConsumerAutoscaler 是 Pod 横向自动扩展的 spec，用于 flowlogs-pipeline-transformer ，它使用 Kafka 信息。当 Kafka 被禁用时，会忽略此设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2) 。
kafkaConsumerBatchSize	整数	kafkaConsumerBatchSize 表示代理消费者将接受的最大批处理大小（以字节为单位）。如果不使用 Kafka，则忽略。默认：10MB。
kafkaConsumerQueueCapacity	整数	kafkaConsumerQueueCapacity 定义 Kafka 消费者客户端中使用的内部消息队列的容量。如果不使用 Kafka，则忽略。

属性	类型	描述
kafkaConsumerReplicas	整数	kafkaConsumerReplicas 定义了为 flowlogs-pipeline-transformer 启动的副本数，它使用 Kafka 信息。当 Kafka 被禁用时，会忽略此设置。
logLevel	字符串	处理器运行时的 logLevel
logTypes	字符串	logTypes 定义要生成的记录类型。可能的值有： <ul style="list-style-type: none"> - FLOWS (默认) 导出常规网络流 - CONVERSATIONS，生成用于开始对话的事件、结束对话以及定期的"tick"更新 - ENDED_CONVERSATIONS 以仅生成终止的对话事件 - ALL 以生成网络流和所有对话事件
metrics	object	Metrics 定义有关指标的处理器配置
端口	整数	流收集器（主机端口）的端口。按照惯例，一些值会被禁止。它必须大于 1024，且不能是 4500、4789 和 6081。
profilePort	整数	profilePort 允许设置侦听此端口的 Go pprof profiler
资源	object	resources 是此容器所需的计算资源。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

28.9.1.35. .spec.processor.debug

描述

debug 允许设置流处理器的内部配置。本节专门用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS env vars。用户设置其值会自行承担任何风险。

类型

对象

属性	类型	描述
env	对象 (字符串)	env 允许将自定义环境变量传递给底层组件。对于传递一些非常严格的性能调整选项 (如 GOGC、GOMAXPROCS) 非常有用, 它们不应作为 FlowCollector 描述符的一部分公开, 因为它们仅在边缘调试或支持场景中有用。

28.9.1.36. .spec.processor.kafkaConsumerAutoscaler

描述

kafkaConsumerAutoscaler 是 Pod 横向自动扩展的 spec, 用于 **flowlogs-pipeline-transformer**, 它使用 Kafka 信息。当 Kafka 被禁用时, 会忽略此设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2)。

类型

object

28.9.1.37. .spec.processor.metrics

描述

Metrics 定义有关指标的处理配置

类型

对象

属性	类型	描述
disableAlerts	数组 (字符串)	disableAlerts 是应禁用的警报列表。可能的值是： NetObservNoFlows , 它会在特定时间段内没有观察到流时触发。 NetObservLokiError , 它会在因为 Loki 错误而丢弃流时触发。
ignoreTags	数组 (字符串)	ignoreTags 是标签列表, 用于指定要忽略哪些指标。每个指标都与一个标签列表关联。更多详情包括在 https://github.com/netobserv/network-observability-operator/tree/main/controllers/flowlogspipeline/metrics_definitions 。可用标签有： egress , ingress , flows , bytes , packets , namespaces , nodes , workloads 。

属性	类型	描述
server	object	Prometheus scraper 的指标服务器端点配置

28.9.1.38. .spec.processor.metrics.server

描述

Prometheus scraper 的指标服务器端点配置

类型

对象

属性	类型	描述
端口	整数	prometheus HTTP 端口
tls	object	TLS 配置。

28.9.1.39. .spec.processor.metrics.server.tls

描述

TLS 配置。

类型

对象

属性	类型	描述
provided	object	当 type 设置为 PROVIDED 时，TLS 的配置。
type	字符串	选择 TLS 配置类型： - DISABLED （默认）来为端点配置 TLS。- PROVIDED 以手动提供证书文件和密钥文件。- AUTO 使用注解使用 OpenShift Container Platform 自动生成的证书。

28.9.1.40. .spec.processor.metrics.server.tls.provided

描述

当 **type** 设置为 **PROVIDED** 时，TLS 的配置。

类型

对象

属性	类型	描述
certFile	字符串	certFile 定义配置映射或 secret 中证书文件名的路径
certKey	字符串	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	字符串	包含证书的配置映射或 Secret 的名称
namespace	字符串	包含证书的配置映射或 secret 的命名空间。如果省略，假设与部署 NetObserv 的命名空间相同。如果命名空间不同，则会复制配置映射或 secret，以便可以根据需要挂载它。
type	字符串	证书引用的类型： configmap 或 secret

28.9.1.41. .spec.processor.resources

描述

resources 是此容器所需的计算资源。更多信息：

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型

对象

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

28.10. 网络流格式参考

这些是网络流格式的规格，在内部和将流导出到 Kafka 时使用。

28.10.1. 网络流格式参考

文档分为两个主要类别：**标签**和常规的**字段**。只有在查询 Loki 时，这个区别才重要。这是因为**标签**必须在**流选择器**中使用，这与**字段**不同，。

如果要读取此规格作为 Kafka 导出功能的引用，您必须将所有**标签**和**字段**视为普通的**字段**，并忽略特定于 Loki 的不同。

28.10.1.1. 标签

SrcK8S_Namespace

- 可选 SrcK8S_Namespace:字符串

源命名空间

DstK8S_Namespace

- 可选 DstK8S_Namespace:字符串

目标命名空间

SrcK8S_OwnerName

- 可选 SrcK8S_OwnerName:字符串

源所有者，如 Deployment、StatefulSet 等。

DstK8S_OwnerName

- 可选 DstK8S_OwnerName:字符串

目标所有者，如 Deployment、StatefulSet 等。

FlowDirection

- **FlowDirection** : 请参阅以下部分，*Enumeration: FlowDirection*。

来自节点观察点的流方向

_RecordType

- 可选 _RecordType:RecordType

记录类型：'flowLog' 用于常规的流日志，或 'allConnections', 'newConnection', 'heartbeat', 'endConnection' 用于对话跟踪

28.10.1.2. 字段

SrcAddr

- **SrcAddr:字符串**

源 IP 地址 (ipv4 或 ipv6)

DstAddr

- **DstAddr: 字符串**

目标 IP 地址 (ipv4 或 ipv6)

SrcMac

- **SrcMac: 字符串**

源 MAC 地址

DstMac

- **DstMac: 字符串**

目标 MAC 地址

SrcK8S_Name

- **可选 SrcK8S_Name:字符串**

与 Kubernetes 对象匹配的源名称，如 Pod 名称、服务名称等。

DstK8S_Name

- **可选 DstK8S_Name:字符串**

与 Kubernetes 对象匹配的目的地名称，如 Pod 名称、服务名称等。

SrcK8S_Type

- **可选 SrcK8S_Type:字符串**

源的类型与 Kubernetes 对象匹配，如 Pod、Service 等。

DstK8S_Type

- **可选 DstK8S_Type:字符串**

目标的类型与 Kubernetes 对象匹配，如 Pod 名称、服务名称等。

SrcPort

- **SrcPort:数字**

源端口

DstPort

- **DstPort:数字**

目的地端口

SrcK8S_OwnerType

- **可选 SrcK8S_OwnerType:字符串**

源 Kubernetes 所有者的类型，如 Deployment、StatefulSet 等。

DstK8S_OwnerType

- **可选 DstK8S_OwnerType:字符串**

目标 Kubernetes 所有者的类型，如 Deployment、StatefulSet 等。

SrcK8S_HostIP

- **可选 SrcK8S_HostIP:字符串**

源节点 IP

DstK8S_HostIP

- **可选 DstK8S_HostIP:字符串**

目的地节点 IP

SrcK8S_HostName

- **可选 SrcK8S_HostName:字符串**

源节点名称

DstK8S_HostName

- **可选 DstK8S_HostName:字符串**

目标节点名称

Proto

- **Proto:数字**

L4 协议

Interface

- 可选 Interface: 字符串

网络接口

Packets

- **Packets: 数字**

此流中的数据包数量

Packets_AB

- 可选 **Packets_AB: 数字**

在对话跟踪中，每个对话中 A 到 B 的数据包计数器

Packets_BA

- 可选 **Packets_BA: 数字**

在对话跟踪中，每个对话中 B 到 A 的数据包计数器

Bytes

- **bytes: 数字**

此流中的字节数

Bytes_AB

- 可选 **Bytes_AB: 数字**

在对话跟踪中，每个对话中 A 到 B 的字节计数器

Bytes_BA

- 可选 **Bytes_BA: 数字**

在对话跟踪中，每个对话中 B 到 A 的字节计数器

TimeFlowStartMs

- **TimeFlowStartMs: 数字**

开始此流的时间戳，以毫秒为单位

TimeFlowEndMs

- **TimeFlowEndMs: 数字**

此流的结束时间戳，以毫秒为单位

TimeReceived

- TimeReceived:数字

由流被流收集器接收并处理时的时间戳，以秒为单位

_HashId

- 可选 _HashId:字符串

在对话跟踪中，对话标识符

_IsFirst

- 可选 _IsFirst:字符串

在对话跟踪中，标识第一个流的标记

numFlowLogs

- 可选 numFlowLogs : 数字

在对话跟踪中，每个对话的流日志计数器

28.10.1.3. Enumeration: FlowDirection

入口

- ingress = "0"

来自节点观察点的传入流量

Egress

- egress = "1"

来自节点观察点的传出流量

28.11. NETWORK OBSERVABILITY 故障排除

为了协助对 Network Observability 问题进行故障排除，可以执行一些故障排除操作。

28.11.1. 使用 must-gather 工具

您可以使用 must-gather 工具来收集有关 Network Observability Operator 资源和集群范围资源的信息，如 pod 日志、**FlowCollector** 和 **Webhook** 配置。

流程

1. 进入到要存储 must-gather 数据的目录。

2. 运行以下命令来收集集群范围的 `must-gather` 资源：

```
$ oc adm must-gather
--image-stream=openshift/must-gather \
--image=quay.io/netobserv/must-gather
```

28.11.2. 在 OpenShift Container Platform 控制台中配置网络流量菜单条目

当网络流量菜单条目没有在 OpenShift Container Platform 控制台的 **Observe** 菜单中列出时，在 OpenShift Container Platform 控制台中手动配置网络流量菜单条目。

先决条件

- 已安装 OpenShift Container Platform 版本 4.10 或更高版本。

流程

1. 运行以下命令，检查 `spec.consolePlugin.register` 字段是否已设置为 `true`：

```
$ oc -n netobserv get flowcollector cluster -o yaml
```

输出示例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  consolePlugin:
    register: false
```

2. 可选：通过手动编辑 Console Operator 配置来添加 `netobserv-plugin` 插件：

```
$ oc edit console.operator.openshift.io cluster
```

输出示例

```
...
spec:
  plugins:
  - netobserv-plugin
...
```

3. 可选：运行以下命令，将 `spec.consolePlugin.register` 字段设置为 `true`：

```
$ oc -n netobserv edit flowcollector cluster -o yaml
```

输出示例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
```



```
name: cluster
spec:
  consolePlugin:
    register: true
```

- 运行以下命令，确保控制台 pod 的状态为 **running**：

```
$ oc get pods -n openshift-console -l app=console
```

- 运行以下命令重启控制台 pod：

```
$ oc delete pods -n openshift-console -l app=console
```

- 清除浏览器缓存和历史记录。

- 运行以下命令，检查 Network Observability 插件 pod 的状态：

```
$ oc get pods -n netobserv -l app=netobserv-plugin
```

输出示例

```
NAME                                READY STATUS RESTARTS AGE
netobserv-plugin-68c7bbb9bb-b69q6  1/1   Running 0      21s
```

- 运行以下命令，检查 Network Observability 插件 pod 的日志：

```
$ oc logs -n netobserv -l app=netobserv-plugin
```

输出示例

```
time="2022-12-13T12:06:49Z" level=info msg="Starting netobserv-console-plugin [build
version: , build date: 2022-10-21 15:15] at log level info" module=main
time="2022-12-13T12:06:49Z" level=info msg="listening on https://:9001" module=server
```

28.11.3. 安装 Kafka 后 Flowlogs-Pipeline 不会消耗网络流

如果您首先使用 **deploymentModel: KAFKA** 部署了流收集器，然后部署 Kafka，则流收集器可能无法正确连接到 Kafka。手动重启 flow-pipeline pod，其中 Flowlogs-pipeline 不使用 Kafka 中的网络流。

流程

- 运行以下命令，删除 flow-pipeline pod 来重启它们：

```
$ oc delete pods -n netobserv -l app=flowlogs-pipeline-transformer
```

28.11.4. 无法从 br-int 和 br-ex 接口查看网络流

br-ex' 和 br-int 是 OSI 第 2 层操作的虚拟网桥设备。eBPF 代理分别在 IP 和 TCP 级别（第 3 和 4 层）中工作。当网络流量由物理主机或虚拟 pod 接口处理时，您可以预期 eBPF 代理捕获通过 **br-ex** 和 **br-int** 的网络流量。如果您限制 eBPF 代理网络接口只附加到 **br-ex** 和 **br-int**，则不会看到任何网络流。

手动删除限制网络接口到 **br-int** 和 **br-ex** 的 **interfaces** 或 **excludeInterfaces** 中的部分。

流程

1. 删除 **interfaces**: ['br-int', 'br-ex'] 字段。这允许代理从所有接口获取信息。或者，您可以指定 Layer-3 接口，例如 **eth0**。运行以下命令：

```
$ oc edit -n netobserv flowcollector.yaml -o yaml
```

输出示例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    type: EBPF
  ebpf:
    interfaces: [ 'br-int', 'br-ex' ] 1
```

- 1** 指定网络接口。

28.11.5. Network Observability 控制器管理器 pod 内存不足

您可以通过修补 Cluster Service Version (CSV) 来增加 Network Observability Operator 的内存限值，其中 Network Observability 控制器管理器 pod 内存不足。

流程

1. 运行以下命令来修补 CSV：

```
$ oc -n netobserv patch csv network-observability-operator.v1.0.0 --type='json' -p='[{"op":
"replace",
"path":"/spec/install/spec/deployments/0/spec/template/spec/containers/0/resources/limits/memc
ry", value: "1Gi"}]'
```

输出示例

```
clusterserviceversion.operators.coreos.com/network-observability-operator.v1.0.0 patched
```

2. 运行以下命令来查看更新的 CSV：

```
$ oc -n netobserv get csv network-observability-operator.v1.0.0 -o
jsonpath='{.spec.install.spec.deployments[0].spec.template.spec.containers[0].resources.limits.r
emory}'
1Gi
```