OpenShift Container Platform 4.18

多网络

在 OpenShift Container Platform 中配置和管理多个网络接口和虚拟路由

Last Updated: 2025-10-09

OpenShift Container Platform 4.18 多网络

在 OpenShift Container Platform 中配置和管理多个网络接口和虚拟路由

Legal Notice

Abstract

本文档解释了如何在 OpenShift Container Platform 中设置和管理主要和从属网络，包括虚拟路由和转发。

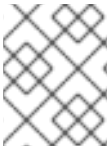
Table of Contents

第 1 章 了解多网络

默认情况下，OVN-Kubernetes 充当 OpenShift Container Platform 集群的 Container Network Interface (CNI)。使用 OVN-Kubernetes 作为集群的默认 CNI，OpenShift Container Platform 管理员或用户可以利用 用户定义的网络(UDN) 或 NetworkAttachmentDefinition (NAD) 创建一个或多个处理集群所有普通网络流量的默认网络。用户定义的网络和网络附加定义可用作以下网络类型：

- **主网络**: Act 作为 pod 的主要网络。默认情况下，所有流量都通过主网络，除非将 pod 路由配置为通过其他网络发送流量。
- **二级网络** ：对 pod 使用额外的非默认网络。二级网络提供专用于特定流量类型或目的的独立接口。只有明确配置为使用二级网络的 pod 流量才会通过其接口路由。

但是，在集群安装过程中，OpenShift Container Platform 管理员可以使用 Multus CNI 插件配置替代的默认二级 pod 网络。使用 Multus 时，可以一起使用多个 CNI 插件，如 ipvlan、macvlan 或 Network Attachment Definitions，以用作 pod 的二级网络。



注意

只有 OVN-Kubernetes 用作 CNI 时，用户定义的网络才可用。它们不支持与其他 CNI 一起使用。

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。

如需支持的 CNI 插件的完整列表，请参阅 "在 OpenShift Container Platform 中添加网络" 。

有关用户定义的网络的详情，请参考关于用户定义的网络(UDN)。

有关网络附加定义的详情，请参考使用 NetworkAttachmentDefinition 创建主网络。

1.1. 二级网络的使用场景

您可以在需要网络隔离的情况下使用二级网络，包括数据平面和控制平面隔离。隔离网络流量对以下性能和安全性原因很有用：

1. 性能
   流量管理：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。
2. 安全性
   网络隔离：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 的二级网络接口，它们名为 **net1**,**net2**, ..., **netN**。

要将二级网络接口附加到 pod，您必须创建配置来定义接口的附加方式。您可以使用 **UserDefinedNetwork** 自定义资源 (CR) 或 **NetworkAttachmentDefinition** CR 指定每个接口。各个 CR 中的 CNI 配置定义如何创建该接口。

有关创建 **UserDefinedNetwork** CR 的更多信息，请参阅关于用户定义的网络。

有关创建 NetworkAttachmentDefinition CR 的更多信息，请参阅使用 NetworkAttach mentDefinition 创建主网络。

1.2. OPENSHIFT CONTAINER PLATFORM 中的二级网络

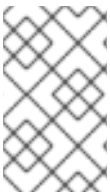
OpenShift Container Platform 为在集群中创建二级网络提供以下 CNI 插件：

- **网桥** ：配置基于网桥的二级网络，以允许同一主机上的 pod 相互通信，并与主机通信。
- **bond-cni** ：配置 Bond CNI 二级网络，以提供将多个网络接口聚合成一个逻辑 *绑定接口* 的方法。
- **host-device** ：配置 host-device 二级网络，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 的二级网络，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 的二级网络不同，每个 pod 共享与父级物理网络接口相同的 MAC 地址。
- **vlan** ：配置基于 VLAN 的二级网络，以为 pod 启用基于 VLAN 的网络隔离和连接。
- **macvlan**：配置基于 macvlan 的二级网络，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 的二级网络的每个 pod 都会获得一个唯一的 MAC 地址。
- **TAP** ：配置基于 TAP 的二级网络，以在容器命名空间内创建 tap 设备。TAP 设备可让用户空间程序发送和接收网络数据包。
- **SR-IOV** ：配置基于 SR-IOV 的二级网络，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。
- **route-override**: 基于二级的网络配置一个 **route-override** 来允许 pod 覆盖和设置路由。

1.3. USERDEFINEDNETWORK 和 NETWORKATTACHMENTDEFINITION 支持列表

UserDefinedNetwork 和 **NetworkAttachmentDefinition** 自定义资源(CR)为集群管理员和用户提供定制网络配置的能力，并定义自己的网络拓扑，确保网络隔离、管理工作负载的 IP 寻址并配置高级网络功能。另外，还提供了第三个 CR **ClusterUserDefinedNetwork**，它允许管理员在集群级别创建和定义跨越多个命名空间的二级网络。

用户定义的网络和网络附加定义可作为主网络接口和二级网络接口，每个都支持 **layer2** 和 **layer3** 拓扑；第三个网络拓扑 Localnet 也支持二级网络的网络附加定义。



注意

自 OpenShift Container Platform 4.18 起，Localnet 拓扑可用于 **UserDefinedNetwork** 和 **ClusterUserDefinedNetwork** CR。它仅适用于利用二级网络的 **NetworkAttachmentDefinition** CR。

以下部分重点介绍了 **UserDefinedNetwork** 和 **NetworkAttachmentDefinition** CR 支持的功能，当它们用作主网络或从属网络时。还包含了 **ClusterUserDefinedNetwork** CR 的一个独立表。

表 1.1. UserDefinedNetwork 和 NetworkAttachmentDefinition CR 的主要网络支持列表

网络功能	Layer2 拓扑	Layer3 拓扑
east-west 流量	✓	✓
north-south 流量	✓	✓
持久性 IP	✓	X
服务	✓	✓
Routes	X	X
EgressIP 资源	✓	✓
多播 [1]	X	✓
NetworkPolicy 资源 [2]	✓	✓
MultinetworkPolicy 资源	X	X

1. 命名空间中必须启用多播，且仅在 OVN-Kubernetes 网络 pod 之间可用。有关多播的更多信息，请参阅"启用项目多播"。

2. 使用主网络类型创建 **UserDefinedNetwork** CR 时，必须在 **UserDefinedNetwork** CR *后*创建网络策略。

表 1.2. UserDefinedNetwork 和 NetworkAttachmentDefinition CR 的二级网络支持列表

网络功能	Layer2 拓扑	Layer3 拓扑	Localnet 拓扑 [1]
east-west 流量	✓	✓	✓ (只限 **NetworkAttachment Definition** CR)
north-south 流量	X	X	✓ (只限 **NetworkAttachment Definition** CR)
持久性 IP	✓	X	✓ (只限 **NetworkAttachment Definition** CR)
服务	X	X	X
Routes	X	X	X

网络功能	Layer2 拓扑	Layer3 拓扑	Localnet 拓扑 [1]
EgressIP 资源	X	X	X
多播	X	X	X
NetworkPolicy 资源	X	X	X
MultinetworkPolicy 资源	✓	✓	✓ (只限 **NetworkAttachment Definition** CR)

1. Localnet 拓扑无法用于 **UserDefinedNetwork** CR。它仅在 **NetworkAttachmentDefinition** CR 的二级网络上支持。

表 1.3. ClusterUserDefinedNetwork CR 支持列表

网络功能	Layer2 拓扑	Layer3 拓扑
east-west 流量	✓	✓
north-south 流量	✓	✓
持久性 IP	✓	X
服务	✓	✓
Routes	X	X
EgressIP 资源	✓	✓
多播 [1]	X	✓
MultinetworkPolicy 资源	X	X
NetworkPolicy 资源 [2]	✓	✓

1. 命名空间中必须启用多播，且仅在 OVN-Kubernetes 网络 pod 之间可用。如需更多信息，请参阅"关于多播"。
2. 使用主网络类型创建 **ClusterUserDefinedNetwork** CR 时，必须在 **UserDefinedNetwork** CR *后*创建网络策略。

其他资源

- 为项目启用多播

第 2 章 主网络

2.1. 关于用户定义的网络

在实现用户定义的网络(UDN)之前，OpenShift Container Platform 的 OVN-Kubernetes CNI 插件仅在主或 *主* 网络上支持第 3 层拓扑。由于 Kubernetes 设计原则：所有 pod 都附加到主网络，所有 pod 都通过其 IP 地址相互通信，并且 pod 间的流量会根据网络策略的限制。

虽然 Kubernetes 设计对简单部署很有用，但此第 3 层拓扑限制自定义主网络段配置，特别是用于现代多租户部署。

UDN 通过启用自定义第 2 层、第 3 层网络段来提高 Kubernetes pod 网络的默认层 3 拓扑的灵活性和分段功能，其中所有这些片段默认被隔离。这些片段充当容器 pod 和使用默认 OVN-Kubernetes CNI 插件的虚拟机的主网络。UDN 启用广泛的网络架构和拓扑，提高网络灵活性、安全性和性能。

注意

在创建用户定义的网络前，使用 **cgroupv1** Linux Control Groups (cgroup)的节点必须从 **cgroupv1** 重新配置为 **cgroupv2**。如需更多信息，请参阅配置 Linux cgroup。

集群管理员可以通过利用 **ClusterUserDefinedNetwork** 自定义资源(CR)，使用 UDN 创建并定义跨集群级别多个命名空间的主或二级网络。另外，集群管理员或集群用户可以使用 UDN 使用 **UserDefinedNetwork** CR 在命名空间级别定义二级网络。

以下章节进一步强调了用户定义的网络的好处和限制、创建 **ClusterUserDefinedNetwork** 或 **UserDefinedNetwork** CR 时的最佳实践、如何创建 CR 以及与部署相关的其他配置详情。

2.1.1. 用户定义的网络的好处

用户定义的网络具有以下优点：

1. 增强了用于安全性的网络隔离
   - **租户隔离** ：命名空间可以有自己的隔离主网络，类似于在 Red Hat OpenStack Platform (RHOSP) 中隔离租户的方式。这通过降低跨租户流量的风险来提高安全性。
2. 网络灵活性
   - **Layer 2 和 layer 3 的支持**: 集群管理员可以将主网络配置为 Layer 2 或 layer 3 网络类型。
3. 简化网络管理
   - **降低网络配置复杂性**：使用用户定义的网络，可以通过对不同网络中的工作负载进行分组来实现隔离。从而消除了对复杂网络策略的需求。
4. 高级功能
   - **一致且可选择的 IP 寻址** ：用户可以在不同的命名空间和集群间指定并重复使用 IP 子网，从而提供一致的网络环境。
   - **支持多个网络**：用户定义的网络功能允许管理员将多个命名空间连接到单个网络，或者为不同的命名空间组创建不同的网络。
5. 简化了从 Red Hat OpenStack Platform (RHOSP) 迁移应用程序的过程
   - **网络奇偶校验**：通过提供类似网络隔离和配置选项，简化了应用程序从 OpenStack 迁移到

- **网络可视化**：通过提供支持网络隔离和配置选项，简化了应用程序从 OpenStack 迁移到 OpenShift Container Platform 的过程。

开发人员和管理员可以创建用户定义的网络，该网络使用自定义资源是命名空间范围。进程概述如下：

1. 管理员使用 **k8s.ovn.org/primary-user-defined-network** 标签为用户定义网络创建一个命名空间。
2. **UserDefinedNetwork** CR 由集群管理员或用户创建。
3. 用户在命名空间中创建 pod。

2.1.2. 用户定义的网络的限制

虽然用户定义的网络 (UDN) 提供了高度可自定义的网络配置选项，但集群管理员和开发人员在实施和管理这些网络时应了解其中的一些限制。在实施 UDN 前请考虑以下限制。

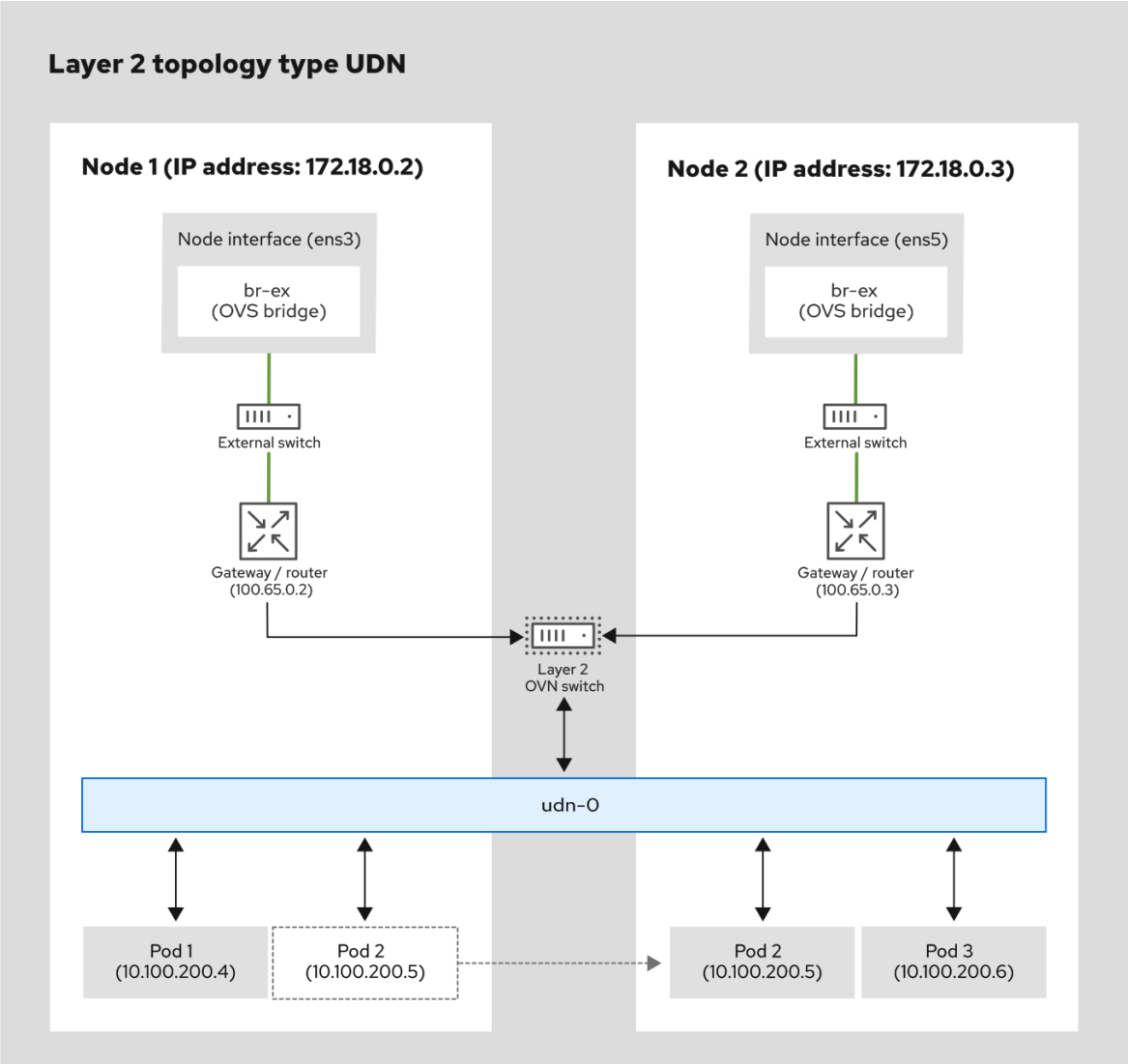
- **DNS 限制** ：
  - DNS 中对 pod 的查询会被解析为集群默认网络上的 pod IP 地址。即使 pod 是用户定义的网络的一部分，DNS 查询也不会解析到用户定义的网络上的 pod IP 地址。但是，服务和外部实体的 DNS 查找将按预期工作。
  - 当 pod 分配给主 UDN 时，它可以访问集群的默认网络上的 Kubernetes API (KAPI) 和 DNS 服务。
- **初始网络分配**：您需要在创建 pod 前创建命名空间和网络。OVN-Kubernetes 不接受将带有 pod 的命名空间分配到一个新的网络，或在现有命名空间中创建一个 UDN。
- **健康检查限制**：Kubelet 健康检查由集群默认网络执行，这不会确认 pod 上主接口的网络连接。因此，在用户定义的网络中可能会出现 pod 在默认网络中是健康的，但在主接口中存在连接问题的情况。
- **网络策略限制** ：启用连接到不同用户定义的主网络的命名空间之间的流量的网络策略无效。这些流量策略不会生效，因为这些隔离的网络之间没有连接。
- **创建和编辑限制** ：在创建后无法修改 **ClusterUserDefinedNetwork** CR 和 **UserDefinedNetwork** CR。
- **默认网络服务访问** ：用户定义的网络 pod 与默认网络隔离，这意味着无法访问大多数默认网络服务。例如，用户定义的网络 pod 目前无法访问 OpenShift Container Platform 镜像 registry。由于这个限制，source-to-image 构建无法在用户定义的网络命名空间中工作。另外，其他功能无法正常工作，包括基于 Git 存储库中的源代码创建应用程序的功能，如 **oc new-app <command>**，以及从使用 source-to-image 构建的 OpenShift Container Platform 模板创建应用程序的功能。这个限制可能会影响其他 **openshift channel.svc** 服务。
- **连接限制** ：用户定义的网络上的 NodePort 服务无法保证隔离。例如，无法从 pod 到同一节点上的服务访问 NodePort 流量，而不同节点上的 pod 的流量会成功。
- **IP 地址耗尽的不明确错误消息** ：当用户定义的网络的子网没有可用 IP 地址时，新 pod 无法启动。当发生这种情况时，会返回以下错误： **Warning: Failed to create pod sandbox**。这个错误消息没有明确地指定 IP 独立性是原因。要确认这个问题，您可以检查 OpenShift Container Platform Web 控制台上 pod 命名空间中的 **Events** 页面，其中报告了有关子网耗尽的明确消息。
- **Layer2 出口 IP 限制** ：
  - 在没有默认网关的情况下，出口 IP 无法正常工作。

- 出口 IP 无法在 Google Cloud Platform (GCP)上工作。
- 出口 IP 不适用于多个网关，而是会将所有流量转发到单个网关。

2.1.3. 第 2 层和第 3 层拓扑

扁平第 2 层拓扑会创建一个虚拟交换机，该交换机分布到集群中的所有节点上。虚拟机和 pod 连接到此虚拟交换机，以便所有这些组件都可以在同一子网内相互通信。扁平第 2 层拓扑可用于在集群中存在的节点间实时迁移虚拟机。下图显示了一个扁平第 2 层拓扑，其中有两个节点使用虚拟交换机进行实时迁移：

图 2.1. 使用虚拟交换机进行组件通信的平面第 2 层拓扑



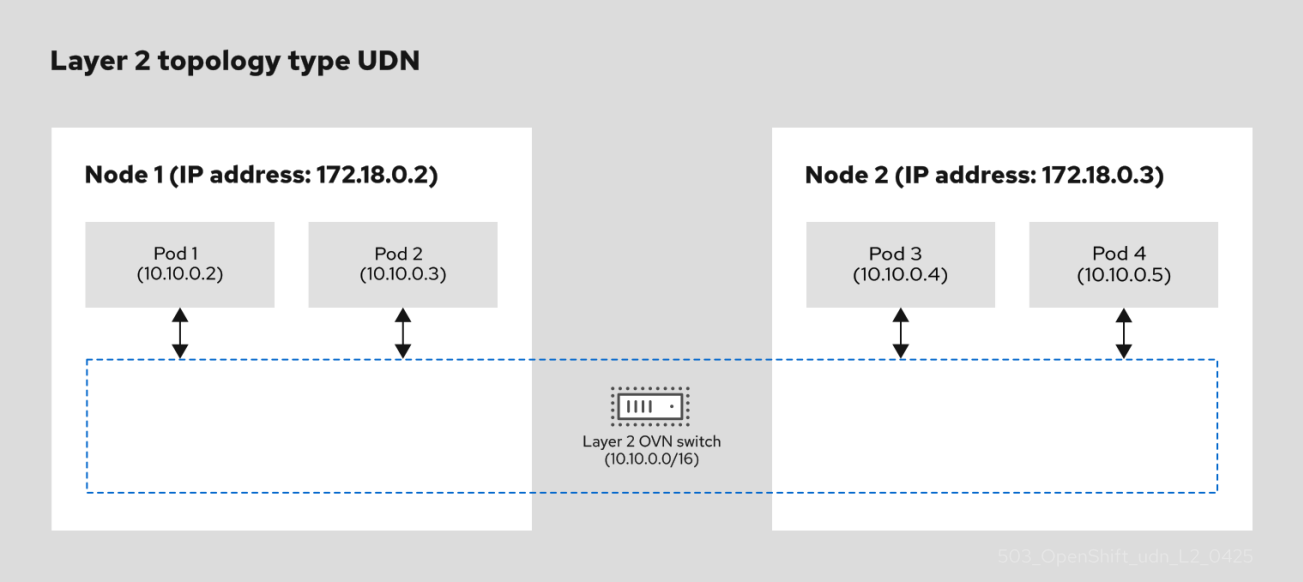
如果您决定不指定第 2 层子网，则必须为集群中的每个 pod 手动配置 IP 地址。当您没有指定第 2 层子网时，端口安全性仅限于防止介质访问控制(MAC)欺骗，且不包含 IP 欺骗。第 2 层拓扑创建了一个在大型网络环境中具有挑战性的广播域，因为拓扑可能会导致广播框架降低网络性能。

要访问更多可配置的选项，您可以将第 2 层拓扑与用户定义的网络(UDN)集成。下图显示了使用 UDN 和第 2 层拓扑的两个节点，其中包含每个节点上存在的 pod。每个节点包含两个接口：

- 节点接口，这是将网络组件连接到节点的计算节点。
- Open vSwitch (OVS)网桥（如 **br-ex** ）创建第 2 层 OVN 交换机，以便 pod 能够相互通信并共享资源。

外部交换机连接这两个接口，而网关或路由器处理外部交换机和第 2 层 OVN 交换机之间的路由流量。节点上的虚拟机和 pod 可以使用 UDN 相互通信。第 2 层 OVN 交换机通过 UDN 处理节点流量，以便将虚拟机从一个节点实时迁移到另一个节点。

图 2.2. 使用第 2 层拓扑的用户定义的网络(UDN)



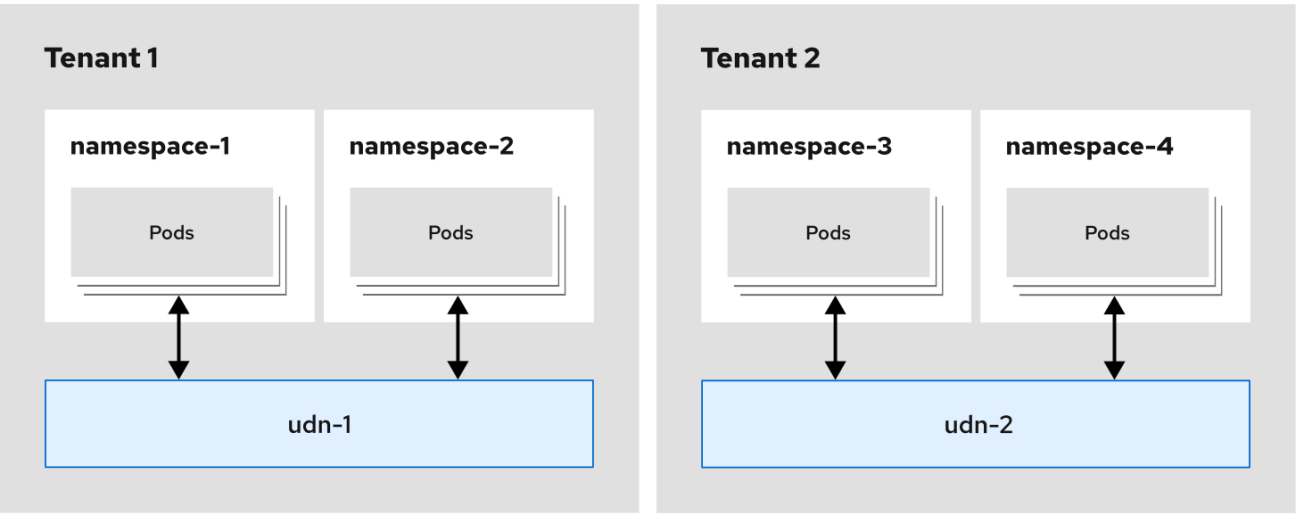
第 3 层拓扑为集群中的每个节点创建一个唯一的第 2 层片段。第 3 层路由机制会连接这些片段，以便在不同节点上托管的虚拟机和 pod 可以相互通信。第 3 层拓扑可以通过将每个域分配给特定节点来有效地管理大型广播域，以便广播流量的范围减少。要配置第 3 层拓扑，您必须配置 **cidr** 和 **hostSubnet** 参数。

2.1.4. 关于 ClusterUserDefinedNetwork CR

ClusterUserDefinedNetwork (UDN)自定义资源(CR)仅为管理员提供集群范围的网络分段和隔离。

下图显示了集群管理员如何使用 **ClusterUserDefinedNetwork** CR 在租户之间创建网络隔离。此网络配置允许网络跨越多个命名空间。在图中，通过创建两个用户定义的网络(**udn-1** 和 **udn** -2)来实现网络隔离。这些网络没有连接，并且 **spec.namespaceSelector.matchLabels** 字段用于选择不同的命名空间。例如，**udn-1** 配置和隔离 **namespace-1** 和 **namespace-2** 的通信，而 **udn-2** 配置和隔离 **namespace-3** 和 **namespace-4**。隔离的租户(Tenants 1 和 Tenants 2)是通过分离命名空间来创建的，同时允许同一命名空间中的 pod 进行通信。

图 2.3. 使用 ClusterUserDefinedNetwork CR 的租户隔离



2.1.4.1. ClusterUserDefinedNetwork CR 的最佳实践

在设置 **ClusterUserDefinedNetwork** 自定义资源(CR)前，用户应考虑以下信息：

- **ClusterUserDefinedNetwork** CR 供集群管理员使用，非管理员用户不应该使用。如果错误地使用，可能会导致部署出现安全问题，从而导致中断或破坏集群网络。
- **ClusterUserDefinedNetwork** CR 不应该选择 **default** 命名空间。这可能导致没有隔离，因此可能会给集群带来安全风险。
- **ClusterUserDefinedNetwork** CR 不应该选择 **openshift channel** 命名空间。
- OpenShift Container Platform 管理员应该注意，当满足以下条件之一时，选择集群的所有命名空间：
  - **matchLabels** 选择器为空。
  - **matchExpressions** 选择器留空。
  - **namespaceSelector** 被初始化，但没有指定 **matchExpressions** 或 **matchLabel**。例如：**namespaceSelector: {}**。
- 对于主网络，用于 **ClusterUserDefinedNetwork** CR 的命名空间必须包含 **k8s.ovn.org/primary-user-defined-network** 标签。此标签无法更新，只能在创建命名空间时添加。以下条件适用于 **k8s.ovn.org/primary-user-defined-network** 命名空间标签：
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签并创建 pod，则 pod 会将自身附加到 default 网络。
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签，并且创建与命名空间匹配的主 **ClusterUserDefinedNetwork** CR，则会报告错误，且不会创建网络。
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签，且主 **ClusterUserDefinedNetwork** CR 已存在，则会创建命名空间中的 pod 并附加到 default 网络。
  - 如果命名空间*有*标签，并且不存在主 **ClusterUserDefinedNetwork** CR，则在创建 **ClusterUserDefinedNetwork** CR 前不会创建命名空间中的 pod。

2.1.4.2. 使用 CLI 创建 ClusterUserDefinedNetwork CR

以下流程使用 CLI 创建 **ClusterUserDefinedNetwork** 自定义资源 (CR)。根据您的用例，使用 **cluster-layer-two-udn.yaml** 示例（**Layer2** 拓扑类型）或 **cluster-layer-three-udn.yaml** 示例（**Layer3** 拓扑类型）创建您的请求。

重要

- **ClusterUserDefinedNetwork** CR 供集群管理员使用，非管理员用户不应该使用。如果错误地使用，可能会导致部署出现安全问题，从而导致中断或破坏集群网络。
- OpenShift Virtualization 只支持 **Layer2** 拓扑。

先决条件

- 您已以具有 **cluster-admin** 权限的用户身份登录。

流程

1. 可选：对于使用主网络的 **ClusterUserDefinedNetwork** CR，请输入以下命令创建带有 **k8s.ovn.org/primary-user-defined-network** 标签的命名空间：

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: <cudn_namespace_name>
  labels:
    k8s.ovn.org/primary-user-defined-network: ""
EOF
```

2. 为 **Layer2** 或 **Layer3** 拓扑类型创建集群范围的用户定义的网络：

   a. 创建 YAML 文件，如 **cluster-layer-two-udn.yaml**，为 **Layer2** 拓扑定义请求，如下例所示：

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: <cudn_name> 1
spec:
  namespaceSelector: 2
    matchLabels: 3
      "<label_1_key>": "<label_1_value>" 4
      "<label_2_key>": "<label_2_value>" 5
  network: 6
    topology: Layer2 7
    layer2: 8
      role: Primary 9
      subnets:
        - "2001:db8::/64"
        - "10.100.0.0/16" 10
```

1. **ClusterUserDefinedNetwork** CR 的名称。
2. CUDN CR 应用到的命名空间集合上的标签查询。使用标准 Kubernetes **MatchLabel** 选择器。不得指向 **default** 或 **openshift channel** 命名空间。
3. 使用 **matchLabels** 选择器类型，其中使用 **AND** 关系评估术语。
4. 5. 在本例中，CUDN CR 被部署到包括 **<label_1_key>=<label_1_value>** 和 **<label_2_key>=<label_2_value>** 标签的命名空间。
6. 描述网络配置。
7. **topology** 字段描述了网络配置；接受的值是 **Layer2** 和 **Layer3**。指定 **Layer2** 拓扑类型会创建一个逻辑交换机，它被所有节点共享。

8 此字段指定拓扑配置。它可以是 **layer2** 或 **layer3**。

9 指定 **Primary** 或 **Secondary**。**Primary** 是 4.18 中唯一支持的 **role** 规格。

10 对于 **Layer2** 拓扑类型，以下指定 **subnet** 字段的配置详情：

- subnets 字段是可选的。
- subnets 字段的类型是**字符串**，接受 IPv4 和 IPv6 的标准 CIDR 格式。
- subnets 字段接受一个或多个项。如果是两个项，它们必须属于不同的系列。例如，子网值 **10.100.0.0/16** 和 **2001:db8::/64**。
- 可以省略 **Layer2** 子网。如果省略，用户必须为 pod 配置静态 IP 地址。因此，端口安全性只能阻止 MAC 欺骗。如需更多信息，请参阅"使用静态 IP 地址配置 pod"。

b. 创建 YAML 文件，如 **cluster-layer-three-udn.yaml**，为 **Layer3** 拓扑定义您的请求，如下例所示：

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: <cudn_name> 1
spec:
  namespaceSelector: 2
    matchExpressions: 3
    - key: kubernetes.io/metadata.name 4
      operator: In 5
      values: ["<example_namespace_one>", "<example_namespace_two>"] 6
  network: 7
    topology: Layer3 8
    layer3: 9
      role: Primary 10
      subnets: 11
        - cidr: 10.100.0.0/16
          hostSubnet: 24
```

1 **ClusterUserDefinedNetwork** CR 的名称。

2 集群 UDN 应用到的命名空间集合上的标签查询。使用标准 Kubernetes **MatchLabel** 选择器。不得指向 **default** 或 **openshift channel** 命名空间。

3 使用 **matchExpressions** 选择器类型，其中使用 **OR** 关系评估术语。

4 指定要匹配的标签键。

5 指定 operator。有效值包括： **In**,**NotIn**,**Exists**, 和 **DoesNotExist**。

6 因为使用了 **matchExpressions** 类型，因此置备命名空间与 **<example_namespace_one>** 或 **<example_namespace_two>** 匹配。

7 描述网络配置。

8. **topology** 字段描述了网络配置；接受的值是 **Layer2** 和 **Layer3**。指定 **Layer3** 拓扑类型会为每个节点创建一个第 2 层段，各自具有不同的子网。第 3 层路由用于互连节点子
9. 此字段指定拓扑配置。有效值为 **layer2** 或 **layer3**。
10. 指定 **Primary** 或 **Secondary** 角色。**Primary** 是 4.18 中唯一支持的 **role** 规格。
11. 对于 **Layer3** 拓扑类型，以下指定 **subnet** 字段的配置详情：
    - **subnets** 字段是必需的。
    - **subnets** 字段的类型是 **cidr** 和 **hostSubnet** ：
      - **cidr** 是集群子网，它接受一个字符串值。
      - **hostSubnet** 指定集群子网分割的节点子网前缀。
      - 对于 IPv6，**hostSubnet** 仅支持 **/64** 长度。

3. 运行以下命令来应用您的请求：

```
$ oc create --validate=true -f <example_cluster_udn>.yaml
```

其中 **<example_cluster_udn>.yaml** 是 **Layer2** 或 **Layer3** 配置文件的名称。

4. 运行以下命令验证您的请求是否成功：

```
$ oc get clusteruserdefinednetwork <cudn_name> -o yaml
```

其中 **<cudn_name>** 是您创建集群范围的用户定义的网络的名称。

输出示例

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  creationTimestamp: "2024-12-05T15:53:00Z"
  finalizers:
  - k8s.ovn.org/user-defined-network-protection
  generation: 1
  name: my-cudn
  resourceVersion: "47985"
  uid: 16ee0fcf-74d1-4826-a6b7-25c737c1a634
spec:
  namespaceSelector:
    matchExpressions:
    - key: custom.network.selector
      operator: In
      values:
      - example-namespace-1
      - example-namespace-2
      - example-namespace-3
  network:
    layer3:
      role: Primary
```

```
    subnets:
    - cidr: 10.100.0.0/16
  topology: Layer3
status:
  conditions:
  - lastTransitionTime: "2024-11-19T16:46:34Z"
    message: 'NetworkAttachmentDefinition has been created in following namespaces:
      [example-namespace-1, example-namespace-2, example-namespace-3]'
    reason: NetworkAttachmentDefinitionReady
    status: "True"
    type: NetworkCreated
```

2.1.4.3. 使用 Web 控制台创建 ClusterUserDefinedNetwork CR

您可以在 OpenShift Container Platform Web 控制台中使用 **Layer2** 拓扑创建 **ClusterUserDefinedNetwork** 自定义资源(CR)。

注意

目前，在使用 OpenShift Container Platform Web 控制台时不支持使用 **Layer3** 拓扑创建 **ClusterUserDefinedNetwork** CR。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform Web 控制台。
- 您已创建了命名空间，并应用 **k8s.ovn.org/primary-user-defined-network** 标签。

流程

1. 从 **Administrator** 视角中，点 **Networking → UserDefinedNetworks**。
2. 点 **ClusterUserDefinedNetwork**。
3. 在 **Name** 字段中，为集群范围的 UDN 指定名称。
4. 在 **Subnet** 字段中指定一个值。
5. 在 **Project (s) Match Labels** 字段中，添加适当的标签以选择集群 UDN 应用到的命名空间。
6. 点 **Create**。集群范围的 UDN 充当位于命名空间中 pod 的默认主网络，其中包含在第 5 步中指定的标签。

其他资源

- 使用静态 IP 地址配置 pod

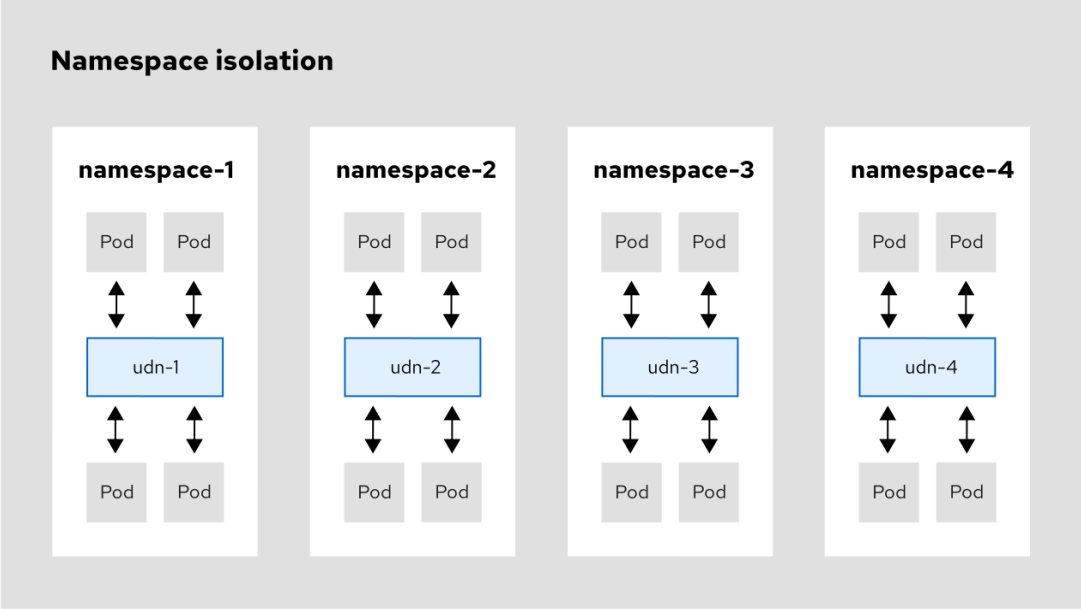
2.1.5. 关于 UserDefinedNetwork CR

UserDefinedNetwork (UDN)自定义资源(CR)为用户和管理员提供高级网络分段和隔离。

下图显示了四个集群命名空间，其中每个命名空间都有一个分配的用户定义的网络(UDN)，每个 UDN 都有一个用于其 pod IP 分配的自定义子网。OVN-Kubernetes 处理任何重叠的 UDN 子网。如果不使用 Kubernetes 网络策略，附加到 UDN 的 pod 可以与 UDN 中的其他 pod 通信。默认情况下，这些 pod 与

其他 UDNs 中存在的 pod 通信。对于微分段，您可以在 UDN 中应用网络策略。您可以将一个或多个 UDN 分配给命名空间，限制为命名空间一个主 UDN，并将一个或多个命名空间分配给 UDN。

图 2.4. 使用 UserDefinedNetwork CR 进行命名空间隔离



2.1.5.1. UserDefinedNetwork CR 的最佳实践

在设置 **UserDefinedNetwork** 自定义资源(CR)前，您应该考虑以下信息：

- **openshift-*** 命名空间不应用于设置 **UserDefinedNetwork** CR。
- **UserDefinedNetwork** CR 不应在 default 命名空间中创建。这可能导致没有隔离，因此可能会给集群带来安全风险。
- 对于主网络，用于 **UserDefinedNetwork** CR 的命名空间必须包含 **k8s.ovn.org/primary-user-defined-network** 标签。此标签无法更新，只能在创建命名空间时添加。以下条件适用于 **k8s.ovn.org/primary-user-defined-network** 命名空间标签：
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签并创建 pod，则 pod 会将自身附加到 default 网络。
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签，并且创建一个与命名空间匹配的主 **UserDefinedNetwork** CR，则会报告状态错误，且不会创建网络。
  - 如果命名空间缺少 **k8s.ovn.org/primary-user-defined-network** 标签和主 **UserDefinedNetwork** CR，则会创建命名空间中的 pod 并附加到 default 网络。
  - 如果命名空间*有*标签，并且不存在主 **UserDefinedNetwork** CR，则在创建 **UserDefinedNetwork** CR 前不会创建命名空间中的 pod。
- 用户定义的网络需要 2 个伪装 IP 地址。您必须重新配置伪装子网，使其足够大，以容纳所需的网络数量。

重要

- 对于 OpenShift Container Platform 4.17 及更新的版本，集群使用 **169.254.0.0/17**（IPv4），或 **fd69::/112**（IPv6）作为默认的伪装子网。用户应避免这些范围。对于更新的集群，默认 masquerade 子网没有变化。
- 在为项目配置了用户定义的网络后，不支持更改集群的伪装子网。在设置 **UserDefinedNetwork** CR 后尝试修改伪装子网可能会破坏网络连接并导致配置问题。

- 确保租户使用 **UserDefinedNetwork** 资源，而不是 **NetworkAttachmentDefinition** (NAD) CR。这可以在租户之间造成安全风险。
- 在创建网络分段时，只有在无法使用 **UserDefinedNetwork** CR 完成用户定义的网络分段时，才应使用 **NetworkAttachmentDefinition** CR。
- **UserDefinedNetwork** CR 的集群子网和服务 CIDR 无法与默认集群子网 CIDR 重叠。OVN-Kubernetes 网络插件使用 **100.64.0.0/16** 作为网络的默认加入子网。您不能使用该值来配置 **UserDefinedNetwork** CR 的 **joinSubnets** 字段。如果在网络中为集群使用默认地址值，则必须通过设置 **joinSubnets** 字段来覆盖默认值。如需更多信息，请参阅"为用户定义的项目添加配置详情"。

2.1.5.2. 使用 CLI 创建 UserDefinedNetwork CR

以下流程创建有命名空间范围的 **UserDefinedNetwork** CR。根据您的用例，创建您的请求，使用 **my-layer-two-udn.yaml** 示例用于 **Layer2** 拓扑类型，或使用 **my-layer-three-udn.yaml** 示例用于 **Layer3** 拓扑类型。

前提条件

- 您已使用 **cluster-admin** 权限登录，或者您有 **view** 和 **edit** 基于角色的访问控制(RBAC)。

流程

1. 可选：对于使用主网络的 **UserDefinedNetwork** CR，请输入以下命令创建带有 **k8s.ovn.org/primary-user-defined-network** 标签的命名空间：

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: <udn_namespace_name>
  labels:
    k8s.ovn.org/primary-user-defined-network: ""
EOF
```

2. 为 **Layer2** 或 **Layer3** 拓扑类型创建用户定义的网络：

   a. 创建一个 YAML 文件，如 **my-layer-two-udn.yaml**，为 **Layer2** 拓扑定义您的请求，如下例所示：

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

```
  name: udn-1 (1)
  namespace: <some_custom_namespace>
spec:
  topology: Layer2 (2)
  layer2: (3)
    role: Primary (4)
    subnets:
      - "10.0.0.0/24"
      - "2001:db8::/60" (5)
```

1. **UserDefinedNetwork** 资源的名称。这不应该是 **default** 或重复 Cluster Network Operator (CNO) 创建的任何全局命名空间。
2. **topology** 字段描述了网络配置；接受的值是 **Layer2** 和 **Layer3**。指定 **Layer2** 拓扑类型会创建一个逻辑交换机，它被所有节点共享。
3. 此字段指定拓扑配置。它可以是 **layer2** 或 **layer3**。
4. 指定 **Primary** 或 **Secondary** 角色。
5. 对于 **Layer2** 拓扑类型，以下指定 **subnet** 字段的配置详情：
   - subnets 字段是可选的。
   - subnets 字段的类型是**字符串**，接受 IPv4 和 IPv6 的标准 CIDR 格式。
   - subnets 字段接受一个或多个项。如果是两个项，它们必须属于不同的系列。例如，子网值 **10.100.0.0/16** 和 **2001:db8::/64**。
   - 可以省略 **Layer2** 子网。如果省略，用户需要为 pod 配置 IP 地址。因此，端口安全性只能阻止 MAC 欺骗。
   - 当指定 **ipamLifecycle** 字段时，**Layer2 subnets** 字段是必须的。

b. 创建一个 YAML 文件，如 **my-layer-three-udn.yaml**，为 **Layer3** 拓扑定义您的请求，如下例所示：

```
apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  name: udn-2-primary (1)
  namespace: <some_custom_namespace>
spec:
  topology: Layer3 (2)
  layer3: (3)
    role: Primary (4)
    subnets: (5)
      - cidr: 10.150.0.0/16
        hostSubnet: 24
      - cidr: 2001:db8::/60
        hostSubnet: 64
# ...
```

1. **UserDefinedNetwork** 资源的名称。这不应该是 **default** 或重复 Cluster Network Operator (CNO) 创建的任何全局命名空间。
2. **topology** 字段描述了网络配置；接受的值是 **Layer2** 和 **Layer3**。指定 **Layer3** 拓扑类型会为每个节点创建一个第 2 层段，各自具有不同的子网。第 3 层路由用于互连节点子网。
3. 此字段指定拓扑配置。有效值为 **layer2** 或 **layer3**。
4. 指定 **Primary** 或 **Secondary** 角色。
5. 对于 **Layer3** 拓扑类型，以下指定 **subnet** 字段的配置详情：
   - **subnets** 字段是必需的。
   - **subnets** 字段的类型是 **cidr** 和 **hostSubnet** ：
     - **CIDR** 等同于集群的 **clusterNetwork** 配置设置。CIDR 中的 IP 地址分布到用户定义的网络中的 pod。此参数接受字符串值。
     - **hostSubnet** 定义每个节点子网的前缀。
     - 对于 IPv6，**hostSubnet** 仅支持 **/64** 长度。

3. 运行以下命令来应用您的请求：

```
$ oc apply -f <my_layer_two_udn>.yaml
```

其中 **<my_layer_two_udn>.yaml** 是 **Layer2** 或 **Layer3** 配置文件的名称。

4. 运行以下命令验证您的请求是否成功：

```
$ oc get userdefinednetworks udn-1 -n <some_custom_namespace> -o yaml
```

其中 **some_custom_namespace** 是您为用户定义的网络创建的命名空间。

输出示例

```
apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  creationTimestamp: "2024-08-28T17:18:47Z"
  finalizers:
  - k8s.ovn.org/user-defined-network-protection
  generation: 1
  name: udn-1
  namespace: some-custom-namespace
  resourceVersion: "53313"
  uid: f483626d-6846-48a1-b88e-6bbeb8bcde8c
spec:
  layer2:
    role: Primary
    subnets:
    - 10.0.0.0/24
    - 2001:db8::/60
```

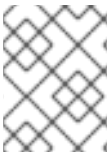
```
  topology: Layer2
status:
  conditions:
  - lastTransitionTime: "2024-08-28T17:18:47Z"
    message: NetworkAttachmentDefinition has been created
    reason: NetworkAttachmentDefinitionReady
    status: "True"
    type: NetworkCreated
```

其他资源

- 默认集群角色

2.1.5.3. 使用 Web 控制台创建 UserDefinedNetwork CR

您可以使用 OpenShift Container Platform Web 控制台创建具有 **Layer2** 拓扑和 **Primary** 角色的 **UserDefinedNetwork** 自定义资源(CR)。



注意

目前，在使用 OpenShift Container Platform Web 控制台时不支持使用 **Layer3** 拓扑或 **Secondary** 角色创建 **UserDefinedNetwork** CR。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问 OpenShift Container Platform Web 控制台。
- 您已创建了命名空间，并应用 **k8s.ovn.org/primary-user-defined-network** 标签。

流程

1. 从 **Administrator** 视角中，点 **Networking → UserDefinedNetworks**。
2. 点 **Create UserDefinedNetwork**。
3. 从 **Project name** 列表中，选择您之前创建的命名空间。
4. 在 **Subnet** 字段中指定一个值。
5. 点 **Create**。用户定义的网络充当您在此命名空间中创建的 pod 的默认主网络。

2.1.6. 用户定义的网络的其他配置详情

下表解释了 **ClusterUserDefinedNetwork** 和 **UserDefinedNetwork** 自定义资源(CR)的额外配置，它们是可选的。不建议在不明确需要和了解 OVN-Kubernetes 网络拓扑的情况下设置这些字段。

1. 用户定义的网络的可选配置

CUDN 字段	UDN 字段	类型	描述

spec.network.<topology>.joinSubnets	**spec.<topology>.joinSubnets**	object	如果省略，平台为 **joinSubnets** 设置默认值 **100.65.0.0/16**（IPv4）和 **fd99::/64**（IPv6）。如果在集群网络的任何位置使用默认地址值，则必须通过设置 **joinSubnets** 字段来覆盖它。如果您选择设置此字段，请确保它与集群中的其他子网没有冲突（如集群子网、**default** 网络集群和 masquerade 子网）。 **joinSubnets** 字段配置一个用户定义的网络中的不同段之间的路由。双栈集群可以设置 2 个子网，每个 IP 系列一个；否则，只设置 1 个子网。此字段仅适用于 **Primary** 网络。
spec.network.<topology>.ipam.lifecycle	**spec.<topology>.ipam.lifecycle**	object	**spec.ipam.lifecycle** 字段配置 IP 地址管理系统(IPAM)。对于虚拟工作负载，您可以使用此字段来确保持久的 IP 地址。唯一允许的值是 **Persistent**，可确保您的虚拟工作负载在重启后具有持久的 IP 地址。由容器网络接口 (CNI) 分配，并由 OVN-Kubernetes 使用来编程 pod IP 地址。对于 pod 注解，不得更改它。 只有在 **ipam.mode** 参数设置为 **Enabled** 时，才支持设置 Persistent 值。

spec.network.<topology>.ipam.mode	**spec.network.<topology>.ipam.mode**	object	**mode** 参数控制 OVN-Kubernetes 管理 IP 配置的数量。可用的选项如下： **Enabled:** 启用后，OVN-Kubernetes 将 IP 配置应用到 SDN 基础架构，并将所选子网的 IP 地址分配给单个 pod。这是默认设置。当设置为 **Enabled** 时，必须定义 **subnets** 字段。**Enabled** 是默认配置。 **Disabled:** 当禁用时，OVN-Kubernetes 只分配 MAC 地址并提供第 2 层通信，允许用户配置 IP 地址。**Disabled** 仅适用于第 2 层（二级）网络。禁用 IPAM 后，通过 IP 依赖 pod 的功能（如网络策略、服务等）不再可以正常工作。另外，附加到这个网络的接口也会禁用 IP 端口安全性。当 **spec.ipam.mode** 设为 **Disabled** 时，**subnets** 字段必须为空。
spec.network.<topology>.mtu	**spec.<topology>.mtu**	整数	最大传输单元 (MTU)。默认值为 **1400**。IPv4 的边界是 **576**，对于 IPv6，是 **1280**。

其中：

<topology>

是 **第 2 层或第 3 层** 之一。

2.1.7. 用户定义的网络状态条件类型

下表解释在描述资源时为 **ClusterUserDefinedNetwork** 和 **UserDefinedNetwork** CR 返回的状态条件类型。这些条件可用于对您的部署进行故障排除。

表 2.1. NetworkCreated 条件类型 (ClusterDefinedNetwork 和 UserDefinedNetwork CR)

状况类型	Status	原因和消息	
NetworkCreated	**True**	为 **True** 时，返回以下原因和消息：	
		原因	**消息**
		NetworkAttachmentDefinitionCreated	'NetworkAttachmentDefinition 在以下命名空间中创建：[example-namespace-1, example-namespace-2, example-namespace-3]'
NetworkCreated	**False**	当 **False** 时，返回以下信息之一：	
		原因	**消息**
		SyncError	**生成 NetworkAttachmentDefinition 失败**
		SyncError	**更新 NetworkAttachmentDefinition 失败**
		SyncError	**命名空间 "<namespace_name>": "<primary_network_name>" 中已存在主网络**
		SyncError	**创建 NetworkAttachmentDefinition 失败：创建 NAD 错误**
		SyncError	**具有所需名称外的 NetworkAttachmentDefinition 已存在**
		SyncError	**将终结器添加到 UserDefinedNetwork 失败**
		NetworkAttachmentDefinitionDeleted	**NetworkAttachmentDefinition 正在被删除： [<namespace>/<nad_name>]**

表 2.2. NetworkAllocationSucceededed 条件类型(**UserDefinedNetwork** CR)

状况类型	Status	原因和消息	
NetworkAllocationSucceeded	**True**	为 **True** 时，返回以下原因和消息：	
		原因	**消息**
		NetworkAllocationSucceeded	**所有同步节点的网络分配成功。**
NetworkAllocationSucceeded	**False**	当 **False** 时，返回以下信息：	

状况类型	Status	原因和消息	
		原因	消息
		InternalError	至少一个节点的网络分配失败：[<node_name>]，检查 UDN 事件以了解更多信息。

2.1.8. 在用户定义的网络 pod 中打开默认网络端口

默认情况下，用户定义的网络上的 pod 与默认网络隔离。这意味着默认网络 pod （如运行监控服务 (Prometheus 或 Alertmanager)或 OpenShift Container Platform 镜像 registry 等无法启动到 UDN pod 的连接。

要允许默认网络 pod 连接到用户定义的网络 pod，您可以使用 **k8s.ovn.org/open-default-ports** 注解。此注解在用户定义的网络 pod 上打开特定的端口，以便从默认网络访问。

以下 pod 规格允许从默认网络端口 **53** 上的端口 **80** 和 UDP 流量进入 TCP 连接：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.ovn.org/open-default-ports: |
      - protocol: tcp
        port: 80
      - protocol: udp
        port: 53
# ...
```

注意

打开端口可在 pod 的默认网络 IP 上访问，而不是其 UDN 网络 IP。

2.2. 使用 NETWORKATTACHMENTDEFINITION 创建主网络

以下小节解释了如何使用 **NetworkAttachmentDefinition** (NAD) 资源创建和管理主网络。

2.2.1. 管理主网络的方法

您可以使用以下两种方法之一管理由 NAD 创建的主网络的生命周期：

- 通过修改 Cluster Network Operator (CNO) 配置。使用此方法时，CNO 会自动创建和管理 **NetworkAttachmentDefinition** 对象。除了管理对象生命周期外，CNO 还可确保 DHCP 可用于使用 DHCP 分配 IP 地址的主网络。
- 通过应用 YAML 清单。使用此方法，您可以通过创建 **NetworkAttachmentDefinition** 对象直接管理主网络。此方法可以调用多个 CNI 插件，以便在 pod 中附加主网络接口。

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主网络。对于任一方法，主网络由您配置的 Container Network Interface(CNI)插件管理。

注意

当使用 OVN SDN 在 Red Hat OpenStack Platform (RHOSP) 中使用多个网络接口部署 OpenShift Container Platform 节点时，二级接口的 DNS 配置可能会优先于主接口的 DNS 配置。在这种情况下，运行以下命令删除附加到二级接口的子网 ID 的 DNS 名称服务器：

```
$ openstack subnet set --dns-nameserver 0.0.0.0 <subnet_id>
```

2.2.2. 使用 Cluster Network Operator 创建主网络附加

Cluster Network Operator (CNO) 管理额外网络定义。当您指定要创建的主网络时，CNO 会自动创建 **NetworkAttachmentDefinition** 自定义资源定义(CRD)。

重要

请勿编辑 Cluster Network Operator 所管理的 **NetworkAttachmentDefinition** CRD。这样做可能会破坏主网络上的网络流量。

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：为主网络创建命名空间：

```
$ 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. 通过为您要创建的主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

- 运行以下命令确认 CNO 创建了 **NetworkAttachmentDefinition** CRD。CNO 创建 CRD 之前可能会有延迟。预期输出显示 NAD CRD 的名称并创建期限（以分钟为单位）。

  ```
  $ 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。

2.2.2.1. 配置主网络附加

主网络通过使用 **k8s.cni.cncf.io** API 组中的 **NetworkAttachmentDefinition** API 来配置。

下表中描述了 API 的配置：

表 2.3. NetworkAttachmentDefinition API 字段

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

2.2.3. 通过应用 YAML 清单来创建主网络附加

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 您在要部署 NAD 的命名空间中工作。

流程

1. 使用主网络配置创建 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",
      "namespace": "namespace2", (1)
      "type": "host-device",
      "device": "eth1",
      "ipam": {
        "type": "dhcp"
      }
    }
```

(1) 可选：您可以指定 NAD 应用到的命名空间。如果您在要部署 NAD 的命名空间中工作，则不需要此规格。

2. 运行以下命令来创建主网络：

```
$ oc apply -f <file>.yaml
```

其中：

<file>

指定包含 YAML 清单的文件名。

第 3 章 二级网络

3.1. 在 OVN-KUBERNETES 上创建二级网络

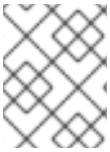
作为集群管理员，您可以使用 **NetworkAttachmentDefinition** (NAD) 资源为集群配置二级网络。

注意

以后的 OpenShift Container Platform 版本中将添加对用户定义的网络作为二级网络的支持。

3.1.1. 配置 OVN-Kubernetes 二级网络

Red Hat OpenShift Networking OVN-Kubernetes 网络插件允许为 pod 配置二级网络接口。要配置二级网络接口，您必须在 **NetworkAttachmentDefinition** 自定义资源定义 (CRD)中定义配置。



注意

Pod 和多网络策略创建可能会处于待处理状态，直到节点的 OVN-Kubernetes control plane 代理处理了关联的 **network-attachment-definition** CRD。

您可以在第 2 层、第 3 层或 localnet 拓扑中配置 OVN-Kubernetes 二级网络。有关这些拓扑支持的功能的更多信息，请参阅"UserDefinedNetwork 和 NetworkAttachmentDefinition 支持列表"。

以下小节提供了 OVN-Kubernetes 当前允许从属网络的每个拓扑配置示例。

注意

网络名称必须是唯一的。例如，不支持使用多个带有不同配置的 **NetworkAttachmentDefinition** CRD。

3.1.1.1. OVN-Kubernetes 二级网络支持的平台

您可以在以下支持的平台中使用 OVN-Kubernetes 二级网络：

- 裸机
- IBM Power®
- IBM Z®
- IBM® LinuxONE
- VMware vSphere
- Red Hat OpenStack Platform(RHOSP)

3.1.1.2. OVN-Kubernetes 网络插件 JSON 配置表

下表描述了 OVN-Kubernetes CNI 网络插件的配置参数：

表 3.1. OVN-Kubernetes 网络插件 JSON 配置表

字段	类型	描述
cniVersion	**string**	CNI 规格版本。所需的值为 **0.3.1**。
name	**string**	网络的名称。这些网络不是命名空间。例如，名为 **l2-network** 的网络可由不同命名空间中的 **NetworkAttachmentDefinition** 自定义资源(CR) 引用。此配置允许在不同命名空间中使用 **NetworkAttachmentDefinition** CR 的 pod 通过同一二级网络进行通信。但是，**NetworkAttachmentDefinition** CR 必须共享相同的特定于网络的参数，如 **topology**,**subnets**,**mtu**,**excludeSubnets**, 和 **vlanID**。只有将 **topology** 字段设置为 **localnet** 时，才应用 **vlanID** 参数。
type	**string**	用于配置的 CNI 插件的名称。这个值必须设置为 **ovn-k8s-cni-overlay**。
topology	**string**	网络的拓扑配置。必须是 **layer2** 或 **localnet** 之一。
subnets	**string**	用于集群间的网络的子网。 对于 **"topology":"layer2"** 部署，支持 IPv6 (**2001:DBB::/64**) 和双栈 (**192.168.100.0/24,2001:DBB::/64**) 子网。 在省略时，实现网络的逻辑交换机仅提供第 2 层通信，用户必须为 pod 配置 IP 地址。端口安全只阻止 MAC 欺骗。
mtu	**string**	最大传输单元 (MTU)。如果没有设置值，Cluster Network Operator (CNO)通过计算主网络接口中的 underlay MTU 的不同来设置默认的 MTU 值，如 pod 网络的 overlay MTU，如 Geneve (Generic Network Virtualization Encapsulation)，以及任何启用的功能（如 IPsec）的字节容量。
netAttachDefName	**string**	包含此配置的网络附加定义 CRD 的元数据 **namespace** 和 **name**。例如，如果在名为 **l2-network** 的命名空间 **ns1** 的 **NetworkAttachmentDefinition** CRD 中定义此配置，则这应设置为 **ns1/l2-network**。
excludeSubnets	**string**	以逗号分隔的 CIDR 和 IP 地址列表。IP 地址从可分配的 IP 地址池中删除，永远不会传递给 pod。
vlanID	**整数**	如果拓扑设置为 **localnet**，则指定的 VLAN 标签将分配给来自这个二级网络的流量。默认为不分配 VLAN 标签。

3.1.1.3. 与多网络策略兼容

多网络策略 API 由 **k8s.cni.cncf.io** API 组中的 **MultiNetworkPolicy** 自定义资源定义(CRD) 提供，与 OVN-Kubernetes 二级网络兼容。在定义网络策略时，可以使用的网络策略规则取决于 OVN-Kubernetes 二级网络是否定义了 **subnets** 字段。详情请查看下表：

表 3.2. 支持基于 subnets CNI 配置的多网络策略选择器

指定的 subnets 字段	允许多网络策略选择器
是	• **podSelector** 和 **namespaceSelector** • **ipBlock**
否	• **ipBlock**

您可以使用 **MultiNetworkPolicy** 对象上的 **k8s.v1.cni.cncf.io/policy-for** 注解指向 **NetworkAttachmentDefinition** (NAD) 自定义资源(CR)。NAD CR 定义策略应用到的网络。只有在名为 **blue2** 的二级网络 CNI 配置中定义了 **subnets** 字段时，以下多网络策略才有效：

使用 pod 选择器的多网络策略示例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: blue2 1
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

以下示例使用 **ipBlock** 网络策略选择器，它始终对 OVN-Kubernetes 二级网络有效：

使用 IP 块选择器的多网络策略示例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name:  ingress-ipblock
  annotations:
    k8s.v1.cni.cncf.io/policy-for: default/flatl2net
spec:
  podSelector:
    matchLabels:
      name: access-control
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.200.0.0/30
```

3.1.1.4. localnet 交换拓扑的配置

交换的 **localnet** 拓扑通过集群范围的逻辑切换到物理网络来连接作为网络附加定义 (NAD) 创建的工作负载。

您必须将二级网络映射到 ovs-bridge，才能将其用作 OVN-Kubernetes 二级网络。网桥映射允许网络流量访问物理网络。网桥映射将物理网络名称（也称为接口标签）与通过 Open vSwitch (OVS)创建的网桥相关联。

您可以创建一个 **NodeNetworkConfigurationPolicy** (NNCP) 对象(**nmstate.io/v1** API 组的一部分)，以声明性地创建映射。此 API 由 NMState Operator 提供。通过使用此 API，您可以将网桥映射应用到与指定 **nodeSelector** 表达式匹配的节点，如 **node-role.kubernetes.io/worker: ''**。使用这个声明方法，NMState Operator 会自动和透明地将二级网络配置应用到节点选择器指定的所有节点。

在附加二级网络时，您可以使用现有的 **br-ex** 网桥或创建新网桥。使用哪种方法取决于您的特定网络基础架构。请考虑以下方法：

- 如果您的节点只包含一个网络接口，则必须使用现有的网桥。这个网络接口由 OVN-Kubernetes 拥有和管理，不得从 **br-ex** 网桥中删除它，或更改接口配置。如果您删除或更改网络接口，您的集群网络将停止工作。
- 如果您的节点包含多个网络接口，您可以将不同的网络接口附加到新网桥，并将该网络接口用于二级网络。这种方法可用于从主集群网络进行流量隔离。

localnet1 网络在以下示例中映射到 **br-ex** 网桥：

共享网桥的映射示例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: mapping 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    ovn:
      bridge-mappings:
      - localnet: localnet1 3
        bridge: br-ex 4
        state: present 5
```

1 1 配置对象的名称。

2 节点选择器指定要将节点网络配置策略应用到的节点。

3 流量转发到 OVS 网桥的二级网络的名称。此二级网络必须与定义 OVN-Kubernetes 额外网络的 **NetworkAttachmentDefinition** CRD 的 **spec.config.name** 字段的名称匹配。

4 节点上的 OVS 网桥的名称。只有在指定 **state: present** 时，才需要这个值。

5 映射的状态。需要是 **present**（添加网桥）或 **absent**（删除网桥）。默认值 **存在**。

以下 JSON 示例配置了一个名为 **localnet1** 的 localnet 二级网络。请注意， **mtu** 参数的值必须与映射到 **br-ex** 网桥接口的二级网络接口设置的 MTU 值匹配。

```
{
```

```
  "cniVersion": "0.3.1",
  "name": "localnet1",
  "type": "ovn-k8s-cni-overlay",
  "topology":"localnet",
  "physicalNetworkName": "localnet1",
  "subnets": "202.10.130.112/28",
  "vlanID": 33,
  "mtu": 1500,
  "netAttachDefName": "ns1/localnet-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

在以下示例中，**localnet2** 网络接口连接到 **ovs-br1** 网桥。通过此附加，网络接口作为二级网络可用于 OVN-Kubernetes 网络插件。

具有多个接口的节点映射示例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ovs-br1-multiple-networks 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    interfaces:
    - name: ovs-br1 3
      description: |-
        A dedicated OVS bridge with eth1 as a port
        allowing all VLANs and untagged traffic
      type: ovs-bridge
      state: up
      bridge:
        allow-extra-patch-ports: true
        options:
          stp: false
          mcast-snooping-enable: true 4
        port:
        - name: eth1 5
    ovn:
      bridge-mappings:
      - localnet: localnet2 6
        bridge: ovs-br1 7
        state: present 8
```

1. 指定配置对象的名称。
2. 指定用于标识节点网络配置策略应用到的节点的节点选择器。
3. 指定一个新的 OVS 网桥，与 OVN-Kubernetes 用于集群流量的默认网桥分开运行。
4. 指定是否启用多播侦听。启用后，多播侦听可防止网络设备向所有网络成员填充多播流量。默认情况下，OVS 网桥不启用多播侦听。默认值为 **false**。

5 指定主机系统上要与新的 OVS 网桥关联的网络设备。

6 指定将流量转发到 OVS 网桥的二级网络的名称。此名称必须与定义 OVN-Kubernetes 二级网络的 **NetworkAttachmentDefinition** CRD 中的 **spec.config.name** 字段的值匹配。

7 指定节点上的 OVS 网桥的名称。只有在设置了 **state: present** 时，才需要该值。

8 指定映射的状态。有效的值是 **present** - 添加网桥；或 **absent** - 删除网桥。默认值 **存在**。

以下 JSON 示例配置了一个名为 **localnet2** 的 localnet 二级网络：请注意， **mtu** 参数的值必须与为 **eth1** 二级网络接口设置的 MTU 值匹配。

```
{
  "cniVersion": "0.3.1",
  "name": "localnet2",
  "type": "ovn-k8s-cni-overlay",
  "topology":"localnet",
  "physicalNetworkName": "localnet2",
  "subnets": "202.10.130.112/28",
  "vlanID": 33,
  "mtu": 1500,
  "netAttachDefName": "ns1/localnet-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

3.1.1.4.1. 配置第 2 层切换拓扑

交换机（层 2）拓扑网络通过集群范围的逻辑交换机互连工作负载。此配置可用于 IPv6 和双栈部署。

注意

第 2 层切换拓扑网络只允许在集群中的 pod 间传输数据数据包。

以下 JSON 示例配置交换的二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "l2-network",
  "type": "ovn-k8s-cni-overlay",
  "topology":"layer2",
  "subnets": "10.100.200.0/24",
  "mtu": 1300,
  "netAttachDefName": "ns1/l2-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

3.1.1.5. 为二级网络配置 pod

您必须通过 **k8s.v1.cni.cncf.io/networks** 注解来指定二级网络附加。

以下示例置备有两个二级附件的 pod，一个用于本指南中提供的每个附加配置。

```
apiVersion: v1
```

```
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: l2-network
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

3.1.1.6. 使用静态 IP 地址配置 pod

以下示例使用静态 IP 地址置备一个 pod。

注意

- 只有在二级网络附加（命名空间范围的对象）使用第 2 层或 localnet 拓扑时，您可以为 pod 的二级网络附加指定 IP 地址。
- 只有在附加配置没有功能子网时，才能为 pod 指定静态 IP 地址。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "l2-network", (1)
        "mac": "02:03:04:05:06:07", (2)
        "interface": "myiface1", (3)
        "ips": [
          "192.0.2.20/24"
        ] (4)
      }
    ]'
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

1. 网络的名称。这个值在所有 **NetworkAttachmentDefinition** CRD 之间必须是唯一的。
2. 为接口分配的 MAC 地址。
3. 为 pod 创建的网络接口的名称。

4 要分配给网络接口的 IP 地址。

3.2. 使用其他 CNI 插件创建二级网络

以下部分介绍了二级网络的具体配置字段。

3.2.1. 配置桥接二级网络

以下对象描述了 Bridge CNI 插件的配置参数：

表 3.3. bridge CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	用于配置的 CNI 插件的名称：**bridge**。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
bridge	**string**	可选：指定要使用的虚拟网桥名称。如果主机上不存在网桥接口，则进行创建。默认值为 **cni0**。
ipMasq	**布尔值**	可选：设置为 **true**，为离开虚拟网络的流量启用 IP 伪装。所有流量的源 IP 地址都会改写为网桥 IP 地址。如果网桥没有 IP 地址，此设置无效。默认值为 **false**。
isGateway	**布尔值**	可选：设置为 **true**，从而为网桥分配 IP 地址。默认值为 **false**。
isDefaultGateway	**布尔值**	可选：设置为 **true**，将网桥配置为虚拟网络的默认网关。默认值为 **false**。如果 **isDefaultGateway** 设置为 **true**，则 **isGateway** 也会自动设置为 **true**。
forceAddress	**布尔值**	可选：设置为 **true**，以允许将之前分配的 IP 地址分配给虚拟网桥。设置为 **false** 时，如果将来自于重叠子集的 IPv4 地址或者 IPv6 地址分配给虚拟网桥，则会发生错误。默认值为 **false**。
hairpinMode	**布尔值**	可选：设置为 **true**，以允许虚拟网桥通过收到它的虚拟端口将其发回。这个模式也被称为 *反射中继*。默认值为 **false**。
promiscMode	**布尔值**	可选：设置为 **true** 以在网桥上启用混杂模式。默认值为 **false**。
vlan	**string**	可选：指定一个虚拟 LAN (VLAN) 标签作为整数值。默认情况下不分配 VLAN 标签。

字段	类型	描述
preserveDefaultVlan	**string**	可选：指示在连接到网桥的 **veth** 端是否保留默认 vlan。默认值为 true。
vlanTrunk	**list**	可选：分配 VLAN 中继标签。默认值为 **none**。
mtu	**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。
enabledad	**布尔值**	可选：为容器侧 **veth** 启用重复的地址检测。默认值为 **false**。
macspoofchk	**布尔值**	可选：启用 mac spoof 检查，将来自容器的流量限制为接口的 mac 地址。默认值为 **false**。

注意

VLAN 参数在 **veth** 的主机端配置 VLAN 标签，并在网桥接口上启用 **vlan_filtering** 功能。

注意

要为 L2 网络配置 uplink，您必须使用以下命令在 uplink 接口上允许 VLAN：

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

3.2.1.1. Bridge CNI 插件配置示例

以下示例配置了一个名为 **bridge-net** 的二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "bridge-net",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}
```

3.2.2. 配置 Bond CNI 二级网络

Bond Container Network Interface (Bond CNI)允许将多个网络接口聚合到一个逻辑"bonded"接口中，从而增强网络冗余和容错。只有 SR-IOV 虚拟功能(VF)支持使用此插件绑定。

下表描述了 Bond CNI 插件的配置参数：

表 3.4. bond CNI 插件 JSON 配置对象

字段	类型	描述
name	**string**	指定提供给此 CNI 网络附加定义的名称。此名称用于识别和引用容器内的接口。
cniVersion	**string**	CNI 规格版本。
type	**string**	指定要配置的 CNI 插件的名称： **bond**。
miimon	**string**	以毫秒为单位指定地址解析协议(ARP)链路监控频率。此参数定义绑定接口发送 ARP 请求的频率，以检查其聚合接口的可用性。
mtu	**整数**	可选：指定绑定的最大传输单元(MTU)。默认值为 1500。
failOverMac	**整数**	可选：指定绑定的 **failOverMac** 设置。默认值为 0。
模式	**string**	指定绑定策略。
linksInContainer	**布尔值**	可选：指定是否应在绑定启动时直接在容器网络命名空间内创建用于绑定的网络接口。如果为 **false** （默认值），CNI 插件在尝试形成绑定前首先在主机系统上查找这些接口。
links	**object**	指定要绑定的接口。
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。

3.2.2.1. bond CNI 插件配置示例

以下示例配置了一个名为 **bond-net1** 的二级网络：

```
{
 "type": "bond",
 "cniVersion": "0.3.1",
 "name": "bond-net1",
 "mode": "active-backup",
 "failOverMac": 1,
 "linksInContainer": true,
 "miimon": "100",
 "mtu": 1500,
 "links": [
      {"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"
    }
}
```

其他资源

- 从两个 SR-IOV 接口配置绑定接口

3.2.3. 主机设备二级网络配置

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 3.5. 主机 device CNI 插件 JSON 配置对象

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

3.2.3.1. host-device 配置示例

以下示例配置了一个名为 **hostdev-net** 的二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "hostdev-net",
  "type": "host-device",
  "device": "eth1"
}
```

3.2.4. 配置 VLAN 二级网络

以下对象描述了 VLAN, **vlan**, CNI 插件的配置参数：

表 3.6. VLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**vlan**。
master	**string**	与网络附加关联的以太网接口。如果没有指定 **master**，则使用默认网络路由的接口。
vlanId	**整数**	设置 **vlan** 的 ID。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
dns	**整数**	可选：要返回的 DNS 信息。例如，优先排序的 DNS 名称服务器列表。
linkInContainer	**布尔值**	可选：指定 **master** 接口是否在容器网络命名空间中或主网络命名空间。将值设为 **true** 以请求使用容器命名空间 **master** 接口。

重要

具有 **vlan** 配置的 **NetworkAttachmentDefinition** 自定义资源定义(CRD)只能在节点上的单个 pod 上使用，因为 CNI 插件无法在同一 **主**接口上创建多个 **vlanId** 的 **vlan** 子接口。

3.2.4.1. VLAN 配置示例

以下示例演示了一个带有名为 **vlan-net** 的二级网络的 **vlan** 配置：

```
{
  "name": "vlan-net",
  "cniVersion": "0.3.1",
  "type": "vlan",
  "master": "eth0",
  "mtu": 1500,
  "vlanId": 5,
  "linkInContainer": false,
  "ipam": {
     "type": "host-local",
     "subnet": "10.1.1.0/24"
  },
  "dns": {
```

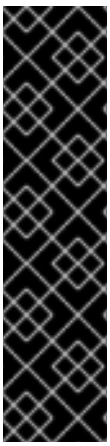
```
    "nameservers": [ "10.1.1.1", "8.8.8.8" ]
  }
}
```

3.2.5. 配置 IPVLAN 二级网络

以下对象描述了 IPVLAN, **ipvlan**, CNI 插件的配置参数：

表 3.7. IPVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**ipvlan**。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。除非插件被串联，否则需要此项。
模式	**string**	可选：虚拟网络的操作模式。这个值必须是 **l2**、**l3** 或 **l3s**。默认值为 **l2**。
master	**string**	可选：与网络附加关联的以太网接口。如果没有指定 **master**，则使用默认网络路由的接口。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
linkInContainer	**布尔值**	可选：指定 **master** 接口是否在容器网络命名空间中或主网络命名空间。将值设为 **true** 以请求使用容器命名空间 **master** 接口。



重要

- **ipvlan** 对象不允许虚拟接口与 **master** 接口通信。因此，容器无法使用 **ipvlan** 接口来访问主机。确保容器加入提供主机连接的网络，如支持 Precision Time Protocol (**PTP**) 的网络。
- 单个 **master** 接口无法同时配置为使用 **macvlan** 和 **ipvlan**。
- 对于不能与接口无关的 IP 分配方案，可以使用处理此逻辑的较早插件来串联 **ipvlan** 插件。如果省略 **master**，则前面的结果必须包含一个接口名称，以便 **ipvlan** 插件进行 enslave。如果省略 **ipam**，则使用前面的结果来配置 **ipvlan** 接口。

3.2.5.1. IPVLAN CNI 插件配置示例

以下示例配置了名为 **ipvlan-net** 的二级网络：

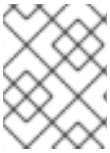
```
{
  "cniVersion": "0.3.1",
  "name": "ipvlan-net",
  "type": "ipvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

3.2.6. 配置 MACVLAN 二级网络

以下对象描述了 MAC Virtual LAN (MACVLAN) Container Network Interface (CNI) 插件的配置参数：

表 3.8. MACVLAN CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	用于配置的 CNI 插件的名称：**macvlan**。
ipam	**object**	IPAM CNI 插件的配置对象。该插件管理附加定义的 IP 地址分配。
模式	**string**	可选：配置虚拟网络上的流量可见性。必须是 **bridge**、**passthru**、**private**或 **Vepa**。如果没有提供值，则默认值为 **bridge**。
master	**string**	可选：与新创建的 macvlan 接口关联的主机网络接口。如果没有指定值，则使用默认路由接口。
mtu	**整数**	可选：将最大传输单元 (MTU) 到指定的值。默认值由内核自动设置。
linkInContainer	**布尔值**	可选：指定 **master** 接口是否在容器网络命名空间中或主网络命名空间。将值设为 **true** 以请求使用容器命名空间 **master** 接口。



注意

如果您为插件配置指定 **master** key，请使用与主网络插件关联的物理网络接口，以避免可能冲突。

3.2.6.1. MACVLAN CNI 插件配置示例

以下示例配置了名为 **macvlan-net** 的二级网络：

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
    }
}
```

3.2.7. 配置 TAP 二级网络

以下对象描述了TAP CNI 插件的配置参数：

表 3.9. TAP CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.3.1** 值。
name	**string**	您之前为 CNO 配置提供的 **name** 参数的值。
type	**string**	要配置的 CNI 插件的名称：**tap**。
mac	**string**	可选：为接口请求指定的 MAC 地址。
mtu	**整数**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
selinuxcontext	**string**	可选：与 tap 设备关联的 SELinux 上下文。  **注意** OpenShift Container Platform 需要 **system_u:system_r:container_t:s0** 的值。
multiQueue	**布尔值**	可选：设置为 **true** 以启用多队列。
owner	**整数**	可选：拥有 tap 设备的用户。

字段	类型	描述
group	**整数**	可选：拥有 tap 设备的组。
bridge	**string**	可选：将 tap 设备设置为已存在的网桥的端口。

3.2.7.1. tap 配置示例

以下示例配置了一个名为 **mynet** 的二级网络：

```
{
 "name": "mynet",
 "cniVersion": "0.3.1",
 "type": "tap",
 "mac": "00:11:22:33:44:55",
 "mtu": 1500,
 "selinuxcontext": "system_u:system_r:container_t:s0",
 "multiQueue": true,
 "owner": 0,
 "group": 0
 "bridge": "br1"
}
```

3.2.7.2. 为TAP CNI 插件设置 SELinux 布尔值

要使用 **container_t** SELinux 上下文创建 tap 设备，请使用 Machine Config Operator (MCO) 在主机上启用 **container_use_devices** 布尔值。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

1. 创建一个新的 YAML 文件，如 **setsebool-container-use-devices.yaml**，详情如下：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: setsebool.service
        contents: |
          [Unit]
```

```
Description=Set SELinux boolean for the TAP CNI plugin
Before=kubelet.service

[Service]
Type=oneshot
ExecStart=/usr/sbin/setsebool container_use_devices=on
RemainAfterExit=true

[Install]
WantedBy=multi-user.target graphical.target
```

2. 运行以下命令来创建新的 **MachineConfig** 对象：

```
$ oc apply -f setsebool-container-use-devices.yaml
```

注意

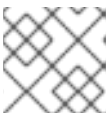
对 **MachineConfig** 对象应用任何更改将导致所有受影响的节点在应用更改后安全重启。此更新可能需要一些时间才能应用。

3. 运行以下命令验证是否应用了更改：

```
$ oc get machineconfigpools
```

预期输出

```
NAME       CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master     rendered-master-e5e0c8e8be9194e7c5a882e047379cfa     True      False
False      3             3                 3                   0                   7d2h
worker     rendered-worker-d6c9ca107fba6cd76cdcbfcedcafa0f2     True      False      False
3             3                 3                   0                   7d
```



注意

所有节点都应处于更新和就绪状态。

3.2.8. 在二级网络上使用 route-override 插件配置路由

以下对象描述了 **route-override** CNI 插件的配置参数：

表 3.10. Route override CNI 插件 JSON 配置对象

字段	类型	描述
type	**string**	要配置的 CNI 插件的名称：**route-override**。
flushroutes	**布尔值**	可选：设置为 **true** 以清除任何现有路由。

字段	类型	描述
flushgateway	**布尔值**	可选：设置为 **true** 以清除网关路由名称。
delroutes	**object**	可选：指定要从容器命名空间中删除的路由列表。
addroutes	**object**	可选：指定要添加到容器命名空间的路由列表。每个路由都是带有 **dst** 和可选的 **gw** 字段的字典。如果省略 **gw**，则插件将使用默认网关值。
skipcheck	**布尔值**	可选：将其设置为 **true** 以跳过 check 命令。默认情况下，CNI 插件会在容器生命周期内验证网络设置。当使用 **route-override** 动态修改路由时，跳过此检查可确保最终配置反映了更新的路由。

3.2.8.1. route-override 插件配置示例

route-override CNI 是一个 CNI 类型，它旨在与父 CNI 串联时使用。它不能独立操作，依赖于父 CNI 来首先创建网络接口并分配 IP 地址，然后才能修改路由规则。

以下示例配置了一个名为 **mymacvlan** 的二级网络。父 CNI 创建一个附加到 **eth1** 的网络接口，并使用 **host-local** IPAM 在 **192.168.1.0/24** 范围内分配一个 IP 地址。然后，**route-override** CNI 链接到父 CNI，并通过清除现有路由来修改路由规则，删除到 **192.168.0.0/24** 的路由，并使用自定义网关为 **192.168.0.0/24** 添加新路由。

```
{
   "cniVersion": "0.3.0",
   "name": "mymacvlan",
   "plugins": [
      {
         "type": "macvlan",          1
         "master": "eth1",
         "mode": "bridge",
         "ipam": {
            "type": "host-local",
            "subnet": "192.168.1.0/24"
         }
      },
      {
         "type": "route-override",   2
         "flushroutes": true,
         "delroutes": [
            {
               "dst": "192.168.0.0/24"
            }
         ],
         "addroutes": [
            {
               "dst": "192.168.0.0/24",
               "gw": "10.1.254.254"
            }
         ]
```

```
    }
  ]
}
```

1. 父 CNI 创建一个附加到 **eth1** 的网络接口。
2. 串联的 **route-override** CNI 会修改路由规则。

其他资源

- 有关在节点上启用 SELinux 布尔值的更多信息，请参阅设置 SELinux 布尔值。

3.3. 将 POD 附加到二级网络

作为集群用户，您可以将 pod 附加到二级网络。

3.3.1. 将 pod 添加到二级网络

您可以将 pod 添加到二级网络。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，二级网络会附加到 pod。但是，如果 pod 已存在，则无法将二级网络附加到其中。

pod 必须与二级网络位于同一个命名空间中。

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. 要在不自定义的情况下附加二级网络，请使用以下格式添加注解。将 **<network>** 替换为要与 pod 关联的二级网络的名称：

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

1. 要指定多个二级网络，请使用逗号分隔每个网络。逗号之间不可包括空格。如果您多次指定相同的二级网络，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二级网络，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** 对象定义的二级网络的名称。
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** 二级网络：

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
          "name": "ovn-kubernetes",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1 **k8s.v1.cni.cncf.io/network-status** 参数是对象的 JSON 数组。每个对象描述附加到 pod 的二级网络的状态。注解值保存为纯文本值。

3.3.1.1. 指定特定于 pod 的地址和路由选项

将 pod 附加到二级网络时，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 必须与二级网络位于同一个命名空间中。
- 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 添加到二级网络，请完成以下步骤：

1. 编辑 **Pod** 资源定义。如果要编辑现有 **Pod** 资源，请运行以下命令在默认编辑器中编辑其定义。将 **<name>** 替换为要编辑的 **Pod** 资源的名称。

```
$ oc edit pod <name>
```

2. 在 **Pod** 资源定义中，将 **k8s.v1.cni.cncf.io/networks** 参数添加到 pod **metadata** 映射中。**k8s.v1.cni.cncf.io/networks** 接受 JSON 字符串，该字符串除指定附加属性外，还引用 **NetworkAttachmentDefinition** 自定义资源（CR）名称的对象。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]'
# ...
```

其中：

<network>

如以下示例所示，将 替换为 JSON 对象。单引号是必需的。

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
```

其中：

name

name 是与 pod 关联的二级网络的名称。

default-route

default-route 指定了一个网关，当在路由表中没有其它路由条目时使用这个网关。如果指定了多个 **default-route** 键，这将导致 pod 无法成为活跃状态。

默认路由将导致任何没有在其它路由中指定的流量被路由到网关。

重要

将 OpenShift Container Platform 的默认路由设置为默认网络接口以外的接口时，可能会导致应该是 pod 和 pod 间的网络流量被路由到其他接口。

要验证 pod 的路由属性，可使用 **oc** 命令在 pod 中执行 **ip** 命令。

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

注意

您还可以通过 JSON 格式的对象列表中的 **default-route** 键来引用 pod 的 **k8s.v1.cni.cncf.io/network-status** 来查看哪个二级网络已被分配默认路由。

要为 pod 设置静态 IP 地址或 MAC 地址，您可以使用 JSON 格式的注解。这要求您创建允许此功能的网络。这可以在 CNO 的 rawCNIConfig 中指定。

3. 运行以下命令来编辑 CNO CR：

```
$ oc edit networks.operator.openshift.io cluster
```

以下 YAML 描述了 CNO 的配置参数：

Cluster Network Operator YAML 配置

```
name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
type: Raw
```

其中：

name

为您要创建的二级网络附加指定名称。该名称在指定的 **namespace** 中需要是唯一的。

namespace

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。

rawCNIConfig

基于以下模板，以 JSON 格式指定 CNI 插件配置。

以下对象描述了使用 macvlan CNI 插件的静态 MAC 地址和 IP 地址的配置参数：

使用静态 IP 和 MAC 地址的 macvlan CNI 插件 JSON 配置对象

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "plugins": [{ 2
     "type": "macvlan",
     "capabilities": { "ips": true }, 3
     "master": "eth0", 4
     "mode": "bridge",
     "ipam": {
       "type": "static"
     }
    }, {
     "capabilities": { "mac": true }, 5
     "type": "tuning"
    }]
}
```

其中：

name

指定要创建的二级网络附加的名称。该名称在指定的 **namespace** 中需要是唯一的。

plugins

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。

ips

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。

master

指定 macvlan 插件使用的接口。

mac

指定一个请求启用 CNI 插件的静态 MAC 地址功能。

以上网络附加可能会以 JSON 格式的注解引用，同时使用相关的键来指定将哪些静态 IP 和 MAC 地址分配给指定 pod。

使用以下内容编辑 pod：

```
$ oc edit pod <name>
```

使用静态 IP 和 MAC 地址的 macvlan CNI 插件 JSON 配置对象

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "<name>", 1
        "ips": [ "192.0.2.205/24" ], 2
        "mac": "CA:FE:C0:FF:EE:00" 3
      }
    ]'
```

1 1 1 1 使用在创建 **rawCNIConfig** 时提供的 **<name>** 。

2 2 2 2 提供包括子网掩码的 IP 地址。

3 3 3 提供 MAC 地址。

注意

静态 IP 地址和 MAC 地址不需要同时使用，您可以单独使用，也可以一起使用。

4. 要验证一个带有二级网络的 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
```

3.4. 配置多网络策略

管理员可以使用 **MultiNetworkPolicy** API 创建多个网络策略，以管理附加到二级网络的 pod 的流量。例如，您可以创建根据特定端口、IP/范围或标签来允许或拒绝流量的策略。

多网络策略可用于管理集群中二级网络上的流量。这些策略无法管理默认集群网络或用户定义的网络的主网络。

作为集群管理员，您可以为以下任何网络类型配置多网络策略：

- 单根 I/O 虚拟化 (SR-IOV)
- MAC 虚拟局域网 (MacVLAN)
- IP 虚拟局域网 (IPVLAN)
- 通过 SR-IOV 绑定 Container Network Interface (CNI)
- OVN-Kubernetes 二级网络

注意

支持为 SR-IOV 二级网络配置多网络策略，仅支持内核网络接口控制器 (NIC)。SR-IOV 不支持 Data Plane Development Kit (DPDK)应用程序。

3.4.1. 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。

- 您可以使用 **MultiNetworkPolicy** 对象上的 **k8s.v1.cni.cncf.io/policy-for** 注解指向 **NetworkAttachmentDefinition** (NAD) 自定义资源(CR)。NAD CR 定义策略应用到的网络。

  包含 k8s.v1.cni.cncf.io/policy-for 注解的多网络策略示例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
```

  其中：

  <namespace_name>

  指定命名空间名称。

  <network_name>

  指定网络附加定义的名称。

3.4.2. 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。成功输出列出了策略对象的名称以及 补丁 的状态。

```
$ oc patch network.operator.openshift.io cluster --type=merge --patch-file=multinetwork-enable-patch.yaml
```

3.4.3. 在 IPv6 网络中支持多网络策略

ICMPv6 Neighbor Discovery Protocol (NDP) 是一组消息和流程，使设备能够发现和维护有关邻居节点的信息。NDP 在 IPv6 网络中扮演着关键角色，用于处理同一链路上的设备间的交互。

当 **useMultiNetworkPolicy** 参数被设置为 **true** 时，Cluster Network Operator (CNO) 会部署多网络策略的 iptables 实现。

要在 IPv6 网络中支持多网络策略，Cluster Network Operator 会在受多网络策略影响的每个 pod 中部署以下规则集：

多网络策略自定义规则

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: multi-networkpolicy-custom-rules
  namespace: openshift-multus
data:

  custom-v6-rules.txt: |
    # accept NDP
    -p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT 1
    -p icmpv6 --icmpv6-type neighbor-advertisement -j ACCEPT 2
    # accept RA/RS
    -p icmpv6 --icmpv6-type router-solicitation -j ACCEPT 3
    -p icmpv6 --icmpv6-type router-advertisement -j ACCEPT 4
```

1. 此规则允许传入的 ICMPv6 邻居请求消息，它们是邻居发现协议 (NDP) 的一部分。这些消息帮助确定邻居节点的链路层地址。
2. 此规则允许传入 ICMPv6 邻居公告消息，它们是 NDP 的一部分，并提供有关发件人链路层地址的信息。
3. 此规则允许传入的 ICMPv6 路由器请求消息。主机使用这些消息来请求路由器配置信息。
4. 此规则允许传入的 ICMPv6 路由器广告消息，为主机提供配置信息。

注意

您不能编辑这些预定义规则。

这些规则共同启用基本的 ICMPv6 流量，以便正确实现网络功能，包括 IPv6 环境中的地址解析和路由器通信。使用这些规则以及多网络策略拒绝流量时，应用程序应不会遇到连接问题。

3.4.4. 使用多网络策略

作为集群管理员，您可以创建、编辑、查看和删除多网络策略。

3.4.4.1. 先决条件

- 您已为集群启用了多网络策略支持。

3.4.4.2. 使用 CLI 创建多网络策略

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

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

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

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

   其中：

   <policy_name>
   指定多网络策略文件名。

   b. 在您刚才创建的文件中定义多网络策略，如下例所示：

   拒绝来自所有命名空间中的所有 pod 的入口流量

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

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

```
  policyTypes:
  - Ingress
  ingress: []
```

其中：

<network_name>

指定网络附加定义的名称。

允许来自所有命名空间中的所有 pod 的入口流量

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

其中：

<network_name>

指定网络附加定义的名称。

允许从特定命名空间中到一个 pod 的入口流量

此策略允许流量从在 **namespace-y** 中运行的 pod 中获取 **pod-a** 标签。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-traffic-pod
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector:
   matchLabels:
      pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

其中：

<network_name>

指定网络附加定义的名称。

限制到服务的流量

应用此策略可确保每个带有标签 **app=bookstore** 和标签 **role=api** 的 pod 只能被带有标签 **app=bookstore** 的 pod 访问。在本例中，应用可以是 REST API 服务器，标记为标签 **app=bookstore** 和 **role=api**。

这个示例可以解决了以下用例：

- 将到一个服务的流量限制为仅使用需要它的其他微服务。
- 将连接限制为只允许使用它的应用程序。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: api-allow
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

其中：

<network_name>

指定网络附加定义的名称。

2. 运行以下命令来创建多网络策略对象。成功输出列出了策略对象的名称 **以及创建** 的状态。

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

其中：

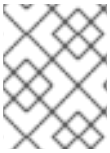
<policy_name>

指定多网络策略文件名。

<namespace>

可选参数。如果您在与当前命名空间不同的命名空间中定义了对象，则参数会特定于命名空间。

成功输出列出了策略对象的名称 **以及创建** 的状态。



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。

3.4.4.3. 编辑多网络策略

您可以编辑命名空间中的多网络策略。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 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

1. 可选： 要列出命名空间中的多网络策略对象，请输入以下命令：

```
$ oc get multi-networkpolicy
```

其中：

<namespace>

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

2. 编辑多网络策略对象。

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

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

其中：

<namespace>

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

<policy_file>

指定包含网络策略的文件的名称。

- 如果您需要直接更新多网络策略对象，请输入以下命令：

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

其中：

<policy_name>

指定网络策略的名称。

<namespace>

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

3. 确认已更新多网络策略对象。

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

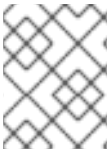
其中：

<policy_name>

指定多网络策略的名称。

<namespace>

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



注意

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

3.4.4.4. 使用 CLI 查看多网络策略

您可以检查命名空间中的多网络策略。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在存在多网络策略的命名空间中工作。

流程

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

```
$ oc get multi-networkpolicy
```

  - 可选： 要检查特定的多网络策略，请输入以下命令：

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

其中：

<policy_name>

指定要检查的多网络策略的名称。

<namespace>

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

注意

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

3.4.4.5. 使用 CLI 删除多网络策略

您可以删除命名空间中的多网络策略。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **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。成功输出列出了策略对象的名称以及 **已删除的** 状态。

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

其中：

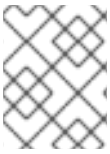
<policy_name>

指定多网络策略的名称。

<namespace>

可选参数。如果您在与当前命名空间不同的命名空间中定义了对象，则参数会特定于命名空间。

成功输出列出了策略对象的名称以及 **已删除的** 状态。



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。

3.4.4.6. 创建默认拒绝所有多网络策略

此策略会阻止由配置其他部署的网络策略和主机网络 pod 间的网络流量允许的所有跨 pod 网络。此流程通过在 **my-project** 命名空间中应用 **deny-by-default** 策略来强制实施强大的拒绝策略。

警告

如果没有配置允许流量通信的 **NetworkPolicy** 自定义资源(CR)，以下策略可能会导致集群中的通信问题。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

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

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  namespace: my-project 1
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name> 2
spec:
  podSelector: {} 3
  policyTypes: 4
  - Ingress 5
  ingress: [] 6
```

1. 指定要部署策略的命名空间。例如，**my-project** 命名空间。
2. 指定命名空间项目的名称，后跟网络附加定义名称。
3. 如果此字段为空，则配置与所有 pod 匹配。因此，该策略适用于 **my-project** 命名空间中的所有 pod。
4. 指定 **NetworkPolicy** 相关规则类型列表。
5. 指定 **Ingress** 仅 **policyTypes**。
6. 指定 **ingress** 规则。如果没有指定，则所有传入的流量会所有 pod 都会丢弃。

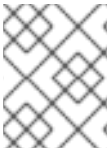
2. 输入以下命令应用策略。成功输出列出了策略对象的名称 **以及创建** 的状态。

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

成功输出列出了策略对象的名称 **以及创建** 的状态。

3.4.4.7. 创建多网络策略以允许来自外部客户端的流量

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



注意

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

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

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

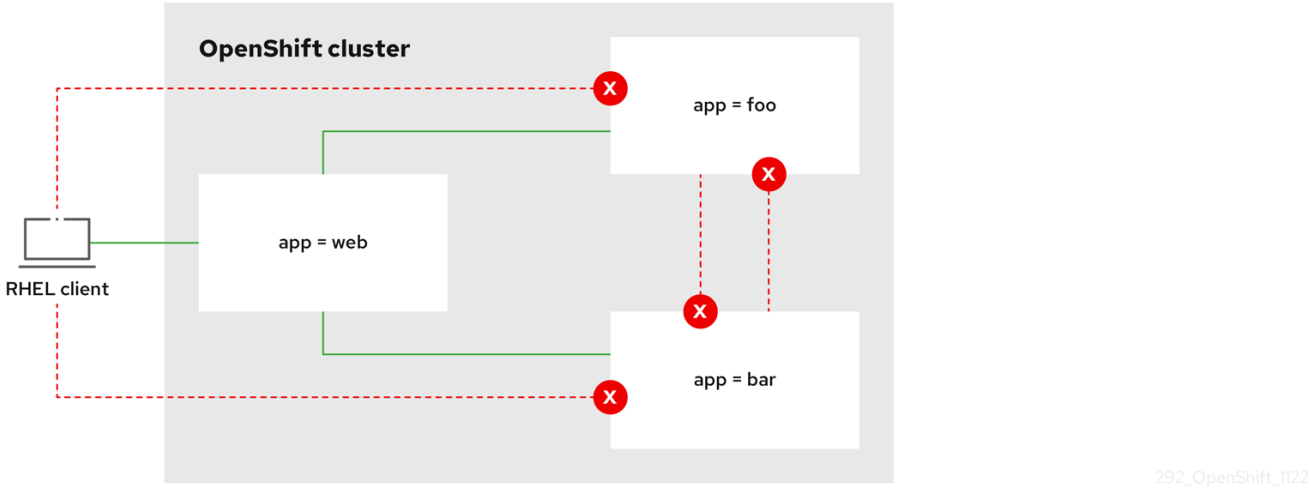
1. 创建策略，以直接从公共互联网的流量或使用负载均衡器访问 pod。将 YAML 保存到 **web-allow-external.yaml** 文件中：

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-external
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

2. 输入以下命令应用策略。成功输出列出了策略对象的名称 **以及创建** 的状态。

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

成功输出列出了策略对象的名称 **以及创建** 的状态。此策略允许来自所有资源的流量，包括下图所示的外部流量：



3.4.4.8. 创建一个多网络策略，允许从所有命名空间中到应用程序的流量

注意

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

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

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

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

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-all-namespaces
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector:
    matchLabels:
```

```
    app: web (1)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {} (2)
```

(1) 仅将策略应用到 default 命名空间中的 **app:web** pod。

(2) 选择所有命名空间中的所有 pod。

注意

默认情况下，如果您没有在策略对象中指定 **namespaceSelector** 参数，则不会选择命名空间。这意味着策略只允许从网络策略部署的命名空间的流量。

2. 输入以下命令应用策略。成功输出列出了策略对象的名称 **以及创建** 的状态。

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

成功输出列出了策略对象的名称 **以及创建** 的状态。

验证

1. 输入以下命令在 **default** 命名空间中启动 web 服务：

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

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

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

3. 在 shell 中运行以下命令，并观察该服务是否允许请求：

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

预期输出

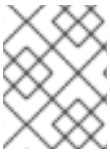
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully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

```
working. Further configuration is required.</p>

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

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

3.4.4.9. 创建多网络策略允许从命名空间中到应用程序的流量



注意

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

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

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

先决条件

- 集群使用支持 **NetworkPolicy** 对象的网络插件，如带有设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络插件。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 权限的用户登陆到集群。
- 您在多网络策略应用到的命名空间中工作。

流程

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

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-prod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
  - Ingress
```

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

1 仅将策略应用到 default 命名空间中的 **app:web** pod。

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

2. 输入以下命令应用策略。成功输出列出了策略对象的名称 **以及创建** 的状态。

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

成功输出列出了策略对象的名称 **以及创建** 的状态。

验证

1. 输入以下命令在 **default** 命名空间中启动 web 服务：

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

2. 运行以下命令来创建 **prod** 命名空间：

```
$ oc create namespace prod
```

3. 运行以下命令来标记 **prod** 命名空间：

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

4. 运行以下命令来创建 **dev** 命名空间：

```
$ oc create namespace dev
```

5. 运行以下命令来标记 **dev** 命名空间：

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

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

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

7. 在 shell 中运行以下命令，并观察请求的原因。例如，预期的输出状态为 **wget：下载超时**。

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

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

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

9. 在 shell 中运行以下命令，并观察是否允许请求：

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

预期输出

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

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

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

3.4.5. 其他资源

- 关于网络策略
- 了解多网络
- 配置 macvlan 网络
- 配置 SR-IOV 网络设备

3.5. 从二级网络中删除 POD

作为集群用户，您可以从二级网络中删除 pod。

3.5.1. 从二级网络中删除 pod

您可以通过删除 pod 来从二级网络中删除 pod。

先决条件

- 二级网络附加到 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。

3.6. 编辑二级网络

作为集群管理员，您可以修改现有二级网络的配置。

3.6.1. 修改二级网络附加定义

作为集群管理员，您可以对现有二级网络进行更改。任何附加到二级网络的 pod 都不会被更新。

先决条件

- 已为集群配置二级网络。
- 安装 OpenShift CLI (**oc**) 。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

要为集群编辑二级网络，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，使用您的更改更新二级网络。
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>** 替换为要显示的二级网络的名称。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"]}} }
```

3.7. 在二级网络中配置 IP 地址分配

以下小节提供了如何为二级网络配置 IP 地址分配的说明和信息。

3.7.1. 配置网络附加的 IP 地址分配

对于辅助网络，您可以使用 IP 地址管理(IPAM) CNI 插件分配 IP 地址，该插件支持各种分配方法，包括动态主机配置协议(DHCP)和静态分配。

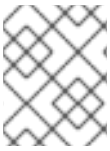
负责动态分配 IP 地址的 DHCP IPAM CNI 插件与两个不同的组件一起运行：

- **CNI 插件** ：负责与 Kubernetes 网络堆栈集成，以请求和释放 IP 地址。
- **DHCP IPAM CNI 守护进程** ：用于 DHCP 事件的监听程序，该事件与环境中的现有 DHCP 服务器协调，以处理 IP 地址分配请求。这个守护进程*并不是* DHCP 服务器本身。

对于在 IPAM 配置中需要 **type: dhcp** 的网络，请确保以下内容：

- DHCP 服务器可用并在环境中运行。
- DHCP 服务器是集群外部的，您希望服务器组成客户的现有网络基础架构的一部分。
- DHCP 服务器被正确配置为为节点提供 IP 地址。

如果环境中 DHCP 服务器不可用，请考虑使用 Whereabouts IPAM CNI 插件。Whereabouts CNI 提供类似的 IP 地址管理功能，而无需外部 DHCP 服务器。



注意

当没有外部 DHCP 服务器或首选静态 IP 地址管理时，请使用 Whereabouts CNI 插件。Whereabouts 插件包含一个协调器守护进程来管理过时的 IP 地址分配。

通过包含单独的守护进程(DHCP IPAM CNI 守护进程)来确保在容器生命周期内定期续订 DHCP 租期。要部署 DHCP IPAM CNI 守护进程，请更改 Cluster Network Operator (CNO)配置，以触发此守护进程的部署，作为二级网络设置的一部分。

3.7.1.1. 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 3.11. ipam 静态配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。值必须是 **static**。
addresses	**数组**	指定分配给虚拟接口的 IP 地址的对象数组。支持 IPv4 和 IPv6 IP 地址。

字段	类型	描述
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 3.12. **ipam.addresses[]** array

字段	类型	描述
address	**string**	您指定的 IP 地址和网络前缀。例如，如果您指定了 **10.10.21.10/24**，二级网络会分配一个 IP 地址 **10.10.21.10**，子网掩码为 **255.255.255.0**。
gateway	**string**	出口网络流量要路由到的默认网关。

表 3.13. **ipam.routes[]** array

字段	类型	描述
dst	**string**	CIDR 格式的 IP 地址范围，如 **192.168.17.0/24** 或默认路由 **0.0.0.0/0**。
gw	**string**	路由网络流量的网关。

表 3.14. **ipam.dns** object

字段	类型	描述
nameservers	**数组**	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"
      }
    ]
  }
}
```

3.7.1.2. 动态 IP 地址(DHCP)分配配置

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。

重要

对于以太网网络附加，SR-IOV Network Operator 不会创建 DHCP 服务器部署。Cluster Network Operator 负责创建最小 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" 1
        }
      }
  # ...
```

1 为集群指定动态 IP 地址(DHCP)分配。

下表描述了使用 DHCP 进行动态 IP 地址地址分配的配置参数。

表 3.15. ipam DHCP 配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。需要值 **dhcp**。

以下 JSON 示例描述了使用 DHCP 进行动态 IP 地址地址分配的配置 p。

动态 IP 地址(DHCP)分配配置示例

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

3.7.1.3. 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 地址分配给二级网络。

Whereabouts CNI 插件还支持在单独的 **NetworkAttachmentDefinition** CRD 中多次出现同一 CIDR 范围的重叠 IP 地址范围和配置。这在多租户环境中提供了更大的灵活性和管理功能。

3.7.1.3.1. 动态 IP 地址配置对象

下表描述了使用 Whereabouts 进行动态 IP 地址分配的配置对象：

表 3.16. ipam whereabouts 配置对象

字段	类型	描述
type	**string**	IPAM 地址类型。需要 **abouts** 的值。
range	**string**	CIDR 表示法中的 IP 地址和范围。IP 地址是通过这个地址范围来分配的。
exclude	**数组**	可选： CIDR 标记中零个或更多 IP 地址和范围的列表。包含在排除地址范围中的 IP 地址。
network_name	**string**	可选：帮助确保每个 pod 的组或域都有自己的一组 IP 地址，即使它们共享相同的 IP 地址范围。设置此字段对于保持网络独立和组织非常重要，特别是在多租户环境中。

3.7.1.3.2. 使用 Whereabouts 的动态 IP 地址分配配置

以下示例显示了使用 Whereabouts 的动态地址分配配置：

Whereabouts 动态 IP 地址分配

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

3.7.1.3.3. 使用 Whereabouts 带有重叠 IP 地址范围的动态 IP 地址分配

以下示例显示了一个动态 IP 地址分配，它将重叠的 IP 地址范围用于多租户网络。

NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 **NetworkAttachmentDefinition 2** 的 **network_name** 匹配。

NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", 1
  }
}
```

1 可选。如果设置，必须与 **NetworkAttachmentDefinition 1** 的 **network_name** 匹配。

3.7.1.4. 创建abouts-reconciler 守护进程集的位置

Whereabouts 协调器负责管理集群中 pod 的动态 IP 地址分配，使用 Whereabouts IP 地址管理 (IPAM) 解决方案。它确保每个 pod 从指定的 IP 地址范围中获取唯一的 IP 地址。它还会在 pod 删除或缩减时处理 IP 地址发行版本。

注意

您还可以使用 **NetworkAttachmentDefinition** 自定义资源定义(CRD)进行动态 IP 地址分配。

当您通过 Cluster Network Operator 配置二级网络时，会自动创建 **whereabouts-reconciler** 守护进程。从 YAML 清单配置二级网络时，它不会自动创建。

要触发 **whereabouts-reconciler** 守护进程集的部署，您必须通过编辑 Cluster Network Operator 自定义资源文件来手动创建一个 **whereabouts-shim** 网络附加。

使用以下步骤部署 **whereabouts-reconciler** 守护进程集。

流程

1. 运行以下命令来编辑 **Network.operator.openshift.io** 自定义资源（CR）：

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

2. 在自定义资源 (CR) 的 **spec** 定义中包含此示例 YAML extract 中显示的 **additionalNetworks** 部分：

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    rawCNIConfig: |-
      {
       "name": "whereabouts-shim",
       "cniVersion": "0.3.1",
       "type": "bridge",
       "ipam": {
         "type": "whereabouts"
       }
      }
    type: Raw
# ...
```

3. 保存文件并退出文本编辑器。

4. 运行以下命令，验证 **whereabouts-reconciler** 守护进程集是否已成功部署：

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

输出示例

```
pod/whereabouts-reconciler-jnp6g 1/1 Running 0 6s
pod/whereabouts-reconciler-k76gg 1/1 Running 0 6s
daemonset.apps/whereabouts-reconciler 6 6 6 6 6 kubernetes.io/os=linux 6s
```

3.7.1.5. 配置 Whereabouts IP 协调器调度

Whereabouts IPAM CNI 插件每天运行 IP 协调器。此过程会清理任何已搁置的 IP 分配，搁置的 IP 分配可能会耗尽 IP 资源，并使新的 pod 无法获取分配给它们的 IP。

使用这个流程更改 IP 协调器运行的频率。

先决条件

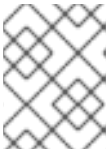
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您已部署了**whereabouts-reconciler** 守护进程集，**whereabouts-reconciler** pod 已启动并正在运行。

流程

1. 运行以下命令，使用 IP 协调器的特定 cron 表达式在 **openshift-multus** 命名空间中创建一个名为 **whereabouts-config** 的 **ConfigMap** 对象：

```
$ oc create configmap whereabouts-config -n openshift-multus --from-
literal=reconciler_cron_expression="*/15 * * * *"
```

此 cron 表达式表示 IP 协调器每 15 分钟运行一次。根据您的特定要求调整表达式。



注意

whereabouts-reconciler 守护进程集只能使用包含五个星号的 cron 表达式模式。目前才不支持带有用于表示秒的第 6 个星号。

2. 运行以下命令，获取与 **openshift-multus** 命名空间中的 **whereabouts-reconciler** 守护进程集和 pod 相关的资源信息。

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

输出示例

```
pod/whereabouts-reconciler-2p7hw                1/1     Running   0          4m14s
pod/whereabouts-reconciler-76jk7                1/1     Running   0          4m14s
daemonset.apps/whereabouts-reconciler   6         6         6       6            6
kubernetes.io/os=linux   4m16s
```

3. 运行以下命令，以验证 **whereabouts-reconciler** pod 是否运行带有配置的时间间隔的 IP 协调器：

```
$ oc -n openshift-multus logs whereabouts-reconciler-2p7hw
```

输出示例

```
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CHMOD
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..data_tmp": RENAME
2024-02-02T16:33:54Z [verbose] using expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] configuration updated to file "/cron-schedule/..data". New
cron expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] successfully updated CRON configuration id "00c2d1c9-
631d-403f-bb86-73ad104a6817" - new cron expression: */15 * * * *
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/config": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_26_17.3874177937": REMOVE
2024-02-02T16:45:00Z [verbose] starting reconciler run
2024-02-02T16:45:00Z [debug] NewReconcileLooper - inferred connection data
2024-02-02T16:45:00Z [debug] listing IP pools
2024-02-02T16:45:00Z [debug] no IP addresses to cleanup
2024-02-02T16:45:00Z [verbose] reconciler success
```

3.7.1.6. 为动态分配双栈 IP 地址创建配置

双栈 IP 地址分配可使用 **ipRanges** 参数进行配置：

- IPv4 地址
- IPv6 地址
- 多个 IP 地址分配

流程

1. 将 **type** 设置为 **whereabouts**。
2. 使用 **ipRanges** 来分配 IP 地址，如下例所示：

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
     {
      "name": "whereabouts-dual-stack",
      "cniVersion": "0.3.1,
      "type": "bridge",
      "ipam": {
        "type": "whereabouts",
        "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
      }
     }
```

3. 将网络附加到 pod。如需更多信息，请参阅"将 pod 添加到二级网络"。
4. 验证是否分配了所有 IP 地址。
5. 运行以下命令，以确保 IP 地址被分配为元数据。

```
$ oc exec -it mypod -- ip a
```

3.8. 关于在容器网络命名空间中配置 MASTER 接口

下面的部分提供了有关如何根据主接口创建和管理 MAC-VLAN、IP-VLAN 和 VLAN 子接口的说明和信息。

3.8.1. 关于在容器网络命名空间中配置 master 接口

您可以创建一个基于容器命名空间中的 **master** 接口的 MAC-VLAN、IP-VLAN 或 VLAN 子接口。您还可以在单独的网络附加定义 CRD 中作为 pod 网络配置的一部分创建 **master** 接口。

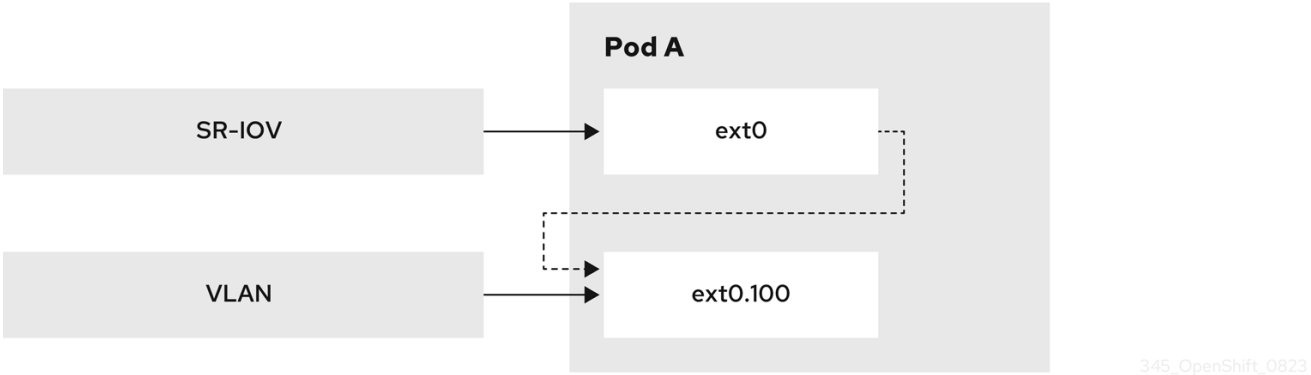
要使用容器命名空间 **master** 接口，您必须为 **NetworkAttachmentDefinition** CRD 的子接口配置中存在的 **linkInContainer** 参数指定 **true**。

3.8.1.1. 在 SR-IOV VF 上创建多个 VLAN

使用此功能的一个用例是基于 SR-IOV VF 的多个 VLAN。要做到这一点，首先创建 SR-IOV 网络，然后为 VLAN 接口定义网络附加。

以下示例演示了如何配置此图中所示的设置。

图 3.1. 创建 VLAN



先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 SR-IOV Network Operator。

流程

1. 使用以下命令，创建您要部署 pod 的专用容器命名空间：

```
$ oc new-project test-namespace
```

2. 创建 SR-IOV 节点策略：

   a. 创建一个 **SriovNetworkNodePolicy** 对象，然后在 **sriov-node-network-policy.yaml** 文件中保存 YAML：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice
 isRdma: false
```

```
needVhostNet: true
nicSelector:
  vendor: "15b3" 1
  deviceID: "101b" 2
  rootDevices: ["00:05.0"]
numVfs: 10
priority: 99
resourceName: sriovnic
nodeSelector:
   feature.node.kubernetes.io/network-sriov.capable: "true"
```

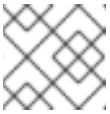
注意

SR-IOV 网络节点策略配置示例，使用设置 **deviceType: netdevice**，专为 Mellanox 网络接口卡(NIC)量身定制。

1. SR-IOV 网络设备厂商的十六进制代码。值 **15b3** 代表与 Mellanox NIC 关联。
2. SR-IOV 网络设备的设备十六进制代码。

b. 运行以下命令来应用 YAML：

```
$ oc apply -f sriov-node-network-policy.yaml
```



注意

应用这可能需要一些时间，因为需要重新引导的节点。

3. 创建 SR-IOV 网络：

a. 为二级 SR-IOV 网络附加创建 **SriovNetwork** 自定义资源(CR)，如下例所示。将 YAML 保存为文件 **sriov-network-attachment.yaml** ：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
 resourceName: sriovnic
 spoofChk: "off"
 trust: "on"
```

b. 运行以下命令来应用 YAML：

```
$ oc apply -f sriov-network-attachment.yaml
```

4. 创建 VLAN 二级网络：

a. 使用以下 YAML 示例，创建一个名为 **vlan100-additional-network-configuration.yaml** 的文件：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: vlan-100
  namespace: test-namespace
spec:
  config: |
    {
      "cniVersion": "0.4.0",
      "name": "vlan-100",
      "plugins": [
        {
          "type": "vlan",
          "master": "ext0", 1
          "mtu": 1500,
          "vlanId": 100,
          "linkInContainer": true, 2
          "ipam": {"type": "whereabouts", "ipRanges": [{"range": "1.1.1.0/24"}]}
        }
      ]
    }
```

1. VLAN 配置需要指定 **master** 名称。这可以在 pod 网络注解中配置。
2. 必须指定 **linkInContainer** 参数。

b. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f vlan100-additional-network-configuration.yaml
```

5. 使用之前指定的网络创建 pod 定义：

a. 使用以下 YAML 示例，创建一个名为 **pod-a.yaml** 文件的文件：

注意

以下清单包括 2 个资源：

- 带有安全标签的命名空间
- 带有适当的网络注解的 Pod 定义

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
---
apiVersion: v1
```

```
kind: Pod
metadata:
  name: nginx-pod
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "sriov-network",
        "namespace": "test-namespace",
        "interface": "ext0" 1
      },
      {
        "name": "vlan-100",
        "namespace": "test-namespace",
        "interface": "ext0.100"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
  containers:
    - name: nginx-container
      image: nginxinc/nginx-unprivileged:latest
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      ports:
        - containerPort: 80
      seccompProfile:
        type: "RuntimeDefault"
```

1 用作 VLAN 接口的 **master** 的名称。

b. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f pod-a.yaml
```

6. 运行以下命令，在 **test-namespace** 中获取 **nginx-pod** 的详细信息：

```
$ oc describe pods nginx-pod -n test-namespace
```

输出示例

```
Name:         nginx-pod
Namespace:    test-namespace
Priority:     0
Node:         worker-1/10.46.186.105
Start Time:   Mon, 14 Aug 2023 16:23:13 -0400
Labels:       <none>
Annotations:  k8s.ovn.org/pod-networks:
                {"default":{"ip_addresses":
["10.131.0.26/23"],"mac_address":"0a:58:0a:83:00:1a","gateway_ips":["10.131.0.1"],"routes":
[{"dest":"10.128.0.0...
```

```
            k8s.v1.cni.cncf.io/network-status:
              [{
                  "name": "ovn-kubernetes",
                  "interface": "eth0",
                  "ips": [
                      "10.131.0.26"
                  ],
                  "mac": "0a:58:0a:83:00:1a",
                  "default": true,
                  "dns": {}
              },{
                  "name": "test-namespace/sriov-network",
                  "interface": "ext0",
                  "mac": "6e:a7:5e:3f:49:1b",
                  "dns": {},
                  "device-info": {
                      "type": "pci",
                      "version": "1.0.0",
                      "pci": {
                          "pci-address": "0000:d8:00.2"
                      }
                  }
              },{
                  "name": "test-namespace/vlan-100",
                  "interface": "ext0.100",
                  "ips": [
                      "1.1.1.1"
                  ],
                  "mac": "6e:a7:5e:3f:49:1b",
                  "dns": {}
              }]
            k8s.v1.cni.cncf.io/networks:
              [ { "name": "sriov-network", "namespace": "test-namespace", "interface": "ext0" }, {
"name": "vlan-100", "namespace": "test-namespace", "i...
            openshift.io/scc: privileged
Status:       Running
IP:           10.131.0.26
IPs:
  IP:  10.131.0.26
```

3.8.1.2. 基于容器命名空间中的网桥主接口创建子接口

您可以根据容器命名空间中的桥接 **master** 接口创建子接口。创建子接口可应用到其他类型的接口。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 以具有 **cluster-admin** 权限的用户身份登录 OpenShift Container Platform 集群。

流程

1. 输入以下命令，创建一个要部署 pod 的专用容器命名空间：

```
$ oc new-project test-namespace
```

2. 使用以下 YAML 示例，创建一个名为 **bridge-nad.yaml** 的桥接 **NetworkAttachmentDefinition** 自定义资源定义 (CRD) 文件：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "bridge-network",
    "type": "bridge",
    "bridge": "br-001",
    "isGateway": true,
    "ipMasq": true,
    "hairpinMode": true,
    "ipam": {
      "type": "host-local",
      "subnet": "10.0.0.0/24",
      "routes": [{"dst": "0.0.0.0/0"}]
    }
  }'
```

3. 运行以下命令，将 **NetworkAttachmentDefinition** CRD 应用到 OpenShift Container Platform 集群：

```
$ oc apply -f bridge-nad.yaml
```

4. 输入以下命令验证您是否已成功创建了 **NetworkAttachmentDefinition** CRD。预期输出显示 NAD CRD 的名称并创建期限（以分钟为单位）。

```
$ oc get network-attachment-definitions
```

5. 使用以下 YAML 示例，为 IPVLAN 二级网络配置创建一个名为 **ipvlan-additional-network-configuration.yaml** 的文件：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ipvlan-net
  namespace: test-namespace
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "ipvlan-net",
    "type": "ipvlan",
    "master": "net1", (1)
    "mode": "l3",
    "linkInContainer": true, (2)
    "ipam": {"type": "whereabouts", "ipRanges": [{"range": "10.0.0.0/24"}]}
  }'
```

(1) 指定要与网络附加关联的以太网接口。这会随后在 pod 网络注解中配置。

2 指定 **master** 接口位于容器网络命名空间中。

6. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f ipvlan-additional-network-configuration.yaml
```

7. 运行以下命令，验证 **NetworkAttachmentDefinition** CRD 是否已成功创建。预期输出显示 NAD CRD 的名称并创建期限（以分钟为单位）。

```
$ oc get network-attachment-definitions
```

8. 使用以下 YAML 示例，为 pod 定义创建一个名为 **pod-a.yaml** 的文件：

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "bridge-network",
        "interface": "net1" 1
      },
      {
        "name": "ipvlan-net",
        "interface": "net2"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-pod
    image: quay.io/openshifttest/hello-
sdn@sha256:c89445416459e7adea9a5a416b3365ed3d74f2491beb904d61dc8d1eb89a72a4

    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1 指定用作 IPVLAN 接口的 **master** 的名称。

9. 运行以下命令来应用 YAML 文件：

```
$ oc apply -f pod-a.yaml
```

10. 使用以下命令验证 pod 是否正在运行：

```
$ oc get pod -n test-namespace
```

输出示例

```
NAME    READY   STATUS    RESTARTS   AGE
pod-a   1/1     Running   0          2m36s
```

11. 运行以下命令，显示 **test-namespace** 中 **pod-a** 资源的网络接口信息：

```
$ oc exec -n test-namespace pod-a -- ip a
```

输出示例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if105: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue
state UP group default
    link/ether 0a:58:0a:d9:00:5d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.217.0.93/23 brd 10.217.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::488b:91ff:fe84:a94b/64 scope link
       valid_lft forever preferred_lft forever
4: net1@if107: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.2/24 brd 10.0.0.255 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fe80::bcda:bdff:fe7e:f437/64 scope link
       valid_lft forever preferred_lft forever
5: net2@net1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global net2
       valid_lft forever preferred_lft forever
    inet6 fe80::beda:bd00:17e:f437/64 scope link
       valid_lft forever preferred_lft forever
```

此输出显示网络接口 **net2** 与物理接口 **net1** 关联。

3.9. 删除额外网络

作为集群管理员，您可以删除额外网络附加。

3.9.1. 删除二级网络附加定义

作为集群管理员，您可以从 OpenShift Container Platform 集群中删除二级网络。二级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

要从集群中删除二级网络，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. 通过删除您要删除的二级网络的 **additionalNetworks** 集合中创建的配置来修改 CR。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] 1
```

1 如果您要删除 **additionalNetworks** 集合中唯一二级网络附加定义的配置映射，您必须指定一个空集合。

3. 要从集群网络中删除网络附加定义，请输入以下命令：

```
$ oc delete net-attach-def <name_of_NAD> 1
```

1 将 **<name_of_NAD>** 替换为网络附加定义的名称。

4. 保存您的更改，再退出文本编辑器以提交更改。
5. 可选：通过运行以下命令确认删除了二级网络 CR：

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

第 4 章 虚拟路由和转发

4.1. 关于虚拟路由和转发

虚拟路由和转发（VRF）设备与 IP 规则相结合，提供了创建虚拟路由和转发域的能力。VRF 减少了 CNF 所需的权限数量，并可提高二级网络网络拓扑的可见性。VRF 用于提供多租户功能，例如，每个租户都有自己的唯一的路由表且需要不同的默认网关。

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

4.1.1. 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。

第 5 章 为 VRF 分配从属网络

作为集群管理员，您可以使用 CNI VRF 插件为虚拟路由和转发 (VRF) 域配置二级网络。此插件创建的虚拟网络与您指定的物理接口关联。

使用带有 VRF 实例的二级网络有以下优点：

工作负载隔离

通过为二级网络配置 VRF 实例来隔离工作负载流量。

提高了安全性

通过 VRF 域中的隔离网络路径启用更高的安全性。

多租户支持

通过网络分段支持每个租户的 VRF 域中唯一路由表的多租户。

注意

使用 VRF 的应用程序必须绑定到特定设备。通常的用法是在套接字中使用 **SO_BINDTODEVICE** 选项。**SO_BINDTODEVICE** 选项将套接字绑定到在传递接口名称中指定的设备，如 **eth1**。要使用 **SO_BINDTODEVICE** 选项，应用程序必须具有 **CAP_NET_RAW** 功能。

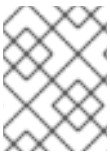
OpenShift Container Platform pod 不支持通过 **ip vrf exec** 命令使用 VRF。要使用 VRF，将应用程序直接绑定到 VRF 接口。

其他资源

- 关于虚拟路由和转发

5.1. 使用 CNI VRF 插件创建二级网络附加

Cluster Network Operator (CNO) 管理二级网络定义。当您指定要创建的二级网络时，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。这样做可能会破坏二级网络上的网络流量。

要使用 CNI VRF 插件创建二级网络附加，请执行以下步骤。

先决条件

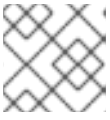
- 安装 OpenShift Container Platform CLI（oc）。
- 以具有 cluster-admin 权限的用户身份登录 OpenShift 集群。

流程

1. 为额外网络附加创建 **Network** 自定义资源 (CR)，并为二级网络插入 **rawCNIConfig** 配置，如下例所示。将 YAML 保存为文件 **additional-network-attachment.yaml**。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: test-network-1
      namespace: additional-network-1
      type: Raw
      rawCNIConfig: '{
        "cniVersion": "0.3.1",
        "name": "macvlan-vrf",
        "plugins": [ 1
        {
          "type": "macvlan",
          "master": "eth1",
          "ipam": {
              "type": "static",
              "addresses": [
              {
                  "address": "191.168.1.23/24"
              }
              ]
          }
        },
        {
          "type": "vrf", 2
          "vrfname": "vrf-1", 3
          "table": 1001 4
        }]
      }'
```

1. **插件**必须是一个列表。列表中的第一个项必须是支持 VRF 网络的从属网络。列表中的第二个项目是 VRF 插件配置。
2. **type** 必须设为 **vrf**。
3. **vrfname** 是接口分配的 VRF 的名称。如果 pod 中不存在，则创建它。
4. 可选。**table** 是路由表 ID。默认情况下使用 **tableid** 参数。如果没有指定，CNI 会为 VRF 分配免费路由表 ID。



注意

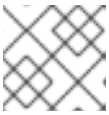
只有在资源类型为 **netdevice** 时，VRF 才能正常工作。

2. 创建 **Network** 资源：

```
$ oc create -f additional-network-attachment.yaml
```

3. 通过运行以下命令确认 CNO 创建了 **NetworkAttachmentDefinition** CR。将 **<namespace>** 替换为您在配置网络附加时指定的命名空间，如 **additional-network-1**。预期输出显示 NAD CR 的名称和创建年龄（以分钟为单位）。

```
$ oc get network-attachment-definitions -n <namespace>
```



注意

CNO 创建 CR 之前可能会有延迟。

验证

1. 创建 pod，并使用 VRF 实例将其分配给二级网络：
   a. 创建定义 **Pod** 资源的 YAML 文件：

   pod-additional-net.yaml 文件示例

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-additional-net
 annotations:
   k8s.v1.cni.cncf.io/networks: '[
      {
            "name": "test-network-1" 1
      }
 ]'
spec:
 containers:
 - name: example-pod-1
   command: ["/bin/bash", "-c", "sleep 9000000"]
   image: centos:8
```

   1 使用 VRF 实例指定二级网络的名称。

   b. 运行以下命令来创建 **Pod** 资源。预期输出显示 **Pod** 资源的名称和创建期限（以分钟为单位）。

```
$ oc create -f pod-additional-net.yaml
```

2. 验证 pod 网络附加是否已连接到 VRF 二级网络。使用 pod 启动远程会话并运行以下命令：预期输出显示 VRF 接口的名称及其在路由表中的唯一 ID。

```
$ ip vrf show
```

3. 确认 VRF 接口是二级接口的控制器：

```
$ ip link
```

输出示例

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
state UP mode
```