Red Hat build of MicroShift 4.16

网络

配置和管理集群网络

Last Updated: 2024-07-02

Red Hat build of MicroShift 4.16 网络

配置和管理集群网络

法律通告

摘要

本文档提供有关配置和管理 MicroShift 集群网络的说明，包括 DNS、ingress 和 Pod 网络。

目录

第 1 章 关于 OVN-KUBERNETES 网络插件

OVN-Kubernetes Container Network Interface (CNI)插件是 MicroShift 集群的默认网络解决方案。OVN-Kubernetes 是 pod 和基于 Open Virtual Network (OVN) 的服务的虚拟网络。

- 默认网络配置和连接会在 MicroShift 中在安装过程中自动应用 **microshift-networking** RPM。
- 使用 OVN-Kubernetes 网络插件的集群也会在节点上运行 Open vSwitch (OVS)。
- OVN-K 在节点上配置 OVS，以实施声明的网络配置。
- 默认情况下，主机物理接口不绑定到 OVN-K 网关网桥 **br-ex**。您可以使用主机上的标准工具来管理默认网关，如 Network Manager CLI (**nmcli**)。
- MicroShift 不支持更改 CNI。

使用配置文件或自定义脚本，您可以配置以下网络设置：

- 您可以使用子网 CIDR 范围为 pod 分配 IP 地址。
- 您可以更改最大传输单元(MTU)值。
- 您可以配置防火墙入口和出口。
- 您可以在 MicroShift 集群中定义网络策略，包括入口和出口规则。
- 您可以使用 MicroShift Multus 插件串联其他 CNI 插件。
- 您可以配置或删除入口路由器。

1.1. MICROSHIFT 网络配置列表

下表总结了作为默认值、配置支持或未通过 MicroShift 服务提供的网络功能和功能的状态：

表 1.1. MicroShift 网络功能和功能概述

网络功能	可用性	支持配置
公告地址	是	是 [1]
Kubernetes 网络策略	是	是
Kubernetes 网络策略日志	不可用	N/A
负载平衡	是	是
多播 DNS	是	是 [2]
网络代理	是 [3]	CRI-O
网络性能	是	MTU 配置

网络功能	可用性	支持配置
出口 IP	不可用	N/A
出口防火墙	不可用	N/A
出口路由器	不可用	N/A
防火墙	没有 [4]	是
硬件卸载	不可用	N/A
混合网络	不可用	N/A
集群内通信的 IPsec 加密	不可用	N/A
IPv6	不可用 [5]	N/A
入口路由器	是	是 [6]
多个网络插件	是	是

1. 如果未设置，则默认值会在服务网络后设置为下一个即时子网。例如，当服务网络为 **10.43.0.0/16** 时，**广告地址** 被设置为 **10.44.0.0/32**。
2. 多播 DNS 协议 (mDNS) 允许使用在 **5353/UDP** 端口上公开的多播进行名称解析和服务发现。
3. MicroShift 中没有内置透明代理的出口流量。出口必须手动配置。
4. RHEL for Edge 支持设置 firewalld 服务。
5. 不支持 IPv6。IPv6 只能通过 MicroShift Multus CNI 插件连接到其他网络来使用。
6. 使用 MicroShift **config.yaml** 文件配置。

1.1.1. 默认设置

如果没有创建 **config.yaml** 文件，则使用默认值。以下示例显示了默认配置设置。

- 要查看默认值，请运行以下命令：

```
$ microshift show-config
```

YAML 格式的默认值示例

```
apiServer:
  advertiseAddress: 10.44.0.0/32 1
  auditLog:
    maxFileAge: 0 2
```

```
    maxFileSize: 200 3
    maxFiles: 10 4
    profile: Default 5
  namedCertificates:
    - certPath: ""
      keyPath: ""
      names:
        - ""
  subjectAltNames: [] 6
debugging:
  logLevel: "Normal" 7
dns:
  baseDomain: microshift.example.com 8
etcd:
  memoryLimitMB: 0 9
ingress:
  listenAddress:
    - "" 10
  ports: 11
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed 12
  status: Managed 13
manifests: 14
  kustomizePaths:
    - /usr/lib/microshift/manifests
    - /usr/lib/microshift/manifests.d/*
    - /etc/microshift/manifests
    - /etc/microshift/manifests.d/*
network:
  clusterNetwork:
    - 10.42.0.0/16 15
  serviceNetwork:
    - 10.43.0.0/16 16
  serviceNodePortRange: 30000-32767 17
node:
  hostnameOverride: "" 18
  nodeIP: "" 19
```

1. 指定 API 服务器公告给集群成员的 IP 地址的字符串。默认值根据服务网络的地址计算。
2. 在自动删除前保留日志文件的时长。**maxFileAge** 参数中的默认值为 **0** 表示日志文件永远不会根据年龄删除。可以配置这个值。
3. 默认情况下，当 **audit.log** 文件达到 **maxFileSize** 限制时，**audit.log** 文件会被轮转，MicroShift 开始写入新的 **audit.log** 文件。可以配置这个值。
4. 保存的日志文件总数。默认情况下，MicroShift 保留 10 个日志文件。创建过量文件时，会删除最旧的文件。可以配置这个值。
5. 仅记录读取和写入请求的日志元数据 ；除了 OAuth 访问令牌请求外，不记录请求正文。如果没有指定此字段，则使用 **Default** 配置集。

6 API 服务器证书的主题备用名称。

7 日志详细程度。此字段的有效值为 **Normal**,**Debug**,**Trace**, 或 **TraceAll**。

8 默认情况下，**etcd** 根据需要使用内存来处理系统上的负载。但是，在内存限制的系统中，可能需要在给定时间限制 **etcd** 可以使用的内存量。

9 集群的基域。所有管理的 DNS 记录都是这个基础的子域。

10 **ingress.listenAddress** 值默认为主机的整个网络。有效可配置的值是一个列表，可以是单个 IP 地址或 NIC 名称，也可以是多个 IP 地址和 NIC 名称。

11 显示的默认端口。可配置。两个端口条目的有效值为 1-65535 范围内的单个唯一端口。**ports.http** 和 **ports.https** 字段的值不能相同。

12 描述如何处理跨命名空间的主机名声明。默认情况下，允许路由在命名空间间声明相同主机名的不同路径。有效值为 **Strict** 和 **InterNamespaceAllowed**。指定 **Strict** 可防止不同命名空间中的路由声明相同的主机名。如果在自定义 MicroShift **config.yaml** 中删除了该值，则会自动设置 **InterNamespaceAllowed** 值。

13 默认路由器状态，可以是 **Managed** 或 **Removed**。

14 用于扫描 **kustomization** 文件的位置，用于加载清单。设置为仅扫描这些路径的路径列表。设置为空列表以禁用加载清单。列表中的条目可以是 glob 模式，以匹配多个子目录。

15 从中分配 Pod IP 地址的 IP 地址块。在安装后此字段是不可变的。

16 Kubernetes 服务的虚拟 IP 地址块。服务的 IP 地址池.支持单个条目。在安装后此字段是不可变的。

17 端口范围允许用于 **NodePort** 类型的 Kubernetes 服务。如果没有指定，则使用默认 30000-32767 范围。没有指定 **NodePort** 的服务会自动从这个范围内分配一个。此参数可以在安装集群后更新。

18 节点的名称。默认值为 hostname。如果非空，则使用此字符串来识别节点，而不是主机名。

19 节点的 IP 地址。默认值是默认路由的 IP 地址。

1.2. 网络功能

MicroShift 4.16 提供的网络功能包括：

- Kubernetes 网络策略
- 动态节点 IP
- 自定义网关接口
- 第二个网关接口
- 指定主机接口上的集群网络
- 阻止对特定主机接口上的 NodePort 服务的外部访问

MicroShift 4.16 不提供网络功能：

- Egress IP/firewall/QoS: disabled
- 混合网络：不支持
- IPsec: 不支持
- 硬件卸载：不支持

1.3. IP 转发

启动时，**ovnkube-master** 容器会自动启用主机网络 **sysctl net.ipv4.ip_forward** 内核参数。这需要将传入的流量转发到 CNI。例如，如果禁用了 **ip_forward**，则从集群外部访问 NodePort 服务会失败。

1.4. 网络性能优化

默认情况下，将三个性能优化应用到 OVS 服务，以最大程度降低资源消耗：

- **ovs-vswitchd.service** 和 **ovsdb-server.service** 的 CPU 关联性
- **no-mlockall** 到 **openvswitch.service**
- 将处理程序和 **revalidator** 线程限制为 **ovs-vswitchd.service**

1.5. MICROSHIFT 网络组件和服务

本简要概述在 MicroShift 中描述了网络组件及其操作。**microshift-networking** RPM 是一个软件包，可自动拉取任何与网络相关的依赖项和 systemd 服务来初始化网络，例如 **microshift-ovs-init** systemd 服务。

NetworkManager

NetworkManager 需要在 MicroShift 节点上设置初始网关网桥。NetworkManager 和 **NetworkManager-ovs** RPM 软件包作为依赖项安装到 **microshift-networking** RPM 软件包，该软件包包含必要的配置文件。MicroShift 中的 NetworkManager 使用 **keyfile** 插件，并在安装 **microshift-networking** RPM 软件包后重新启动。

microshift-ovs-init

microshift-ovs-init.service 由 **microshift-networking** RPM 软件包安装，作为依赖的 systemd 服务到 **microshift.service**。它负责设置 OVS 网关网桥。

OVN 容器

两个 OVN-Kubernetes 守护进程集由 MicroShift 渲染和应用。

- ovnkube-master 包含 **northd**,**nbdb**,**sbdb** 和 **ovnkube-master** 容器。
- **ovnkube-node** ovnkube-node 包含 OVN-Controller 容器。
  MicroShift 启动后，OVN-Kubernetes 守护进程集会在 **openshift-ovn-kubernetes** 命名空间中部署。

打包

OVN-Kubernetes 清单和启动逻辑内置在 MicroShift 中。**microshift-networking** RPM 中包含的 systemd 服务和配置有：

- **/etc/NetworkManager/conf.d/microshift-nm.conf** for **NetworkManager.service**

- **/etc/systemd/system/ovs-vswitchd.service.d/microshift-cpuaffinity.conf** 用于 **ovs-vswitchd.service**
- **/etc/systemd/system/ovsdb-server.service.d/microshift-cpuaffinity.conf** 用于 **ovs-server.service**
- **/usr/bin/configure-ovs-microshift.sh** for **microshift-ovs-init.service**
- **/usr/bin/configure-ovs.sh** for **microshift-ovs-init.service**
- **/etc/crio/crio.conf.d/microshift-ovn.conf** 用于 CRI-O 服务

1.6. 网桥映射

网桥映射允许提供商网络流量访问物理网络。流量离开提供商网络，到达 **br-int** 网桥。**br-int** 和 **br-ex** 之间的跳接端口允许流量遍历提供商网络和边缘网络。Kubernetes pod 通过虚拟以太网对连接到 **br-int** 网桥：一个虚拟以太网对端附加到 pod 命名空间，另一个端点连接到 **br-int** 网桥。

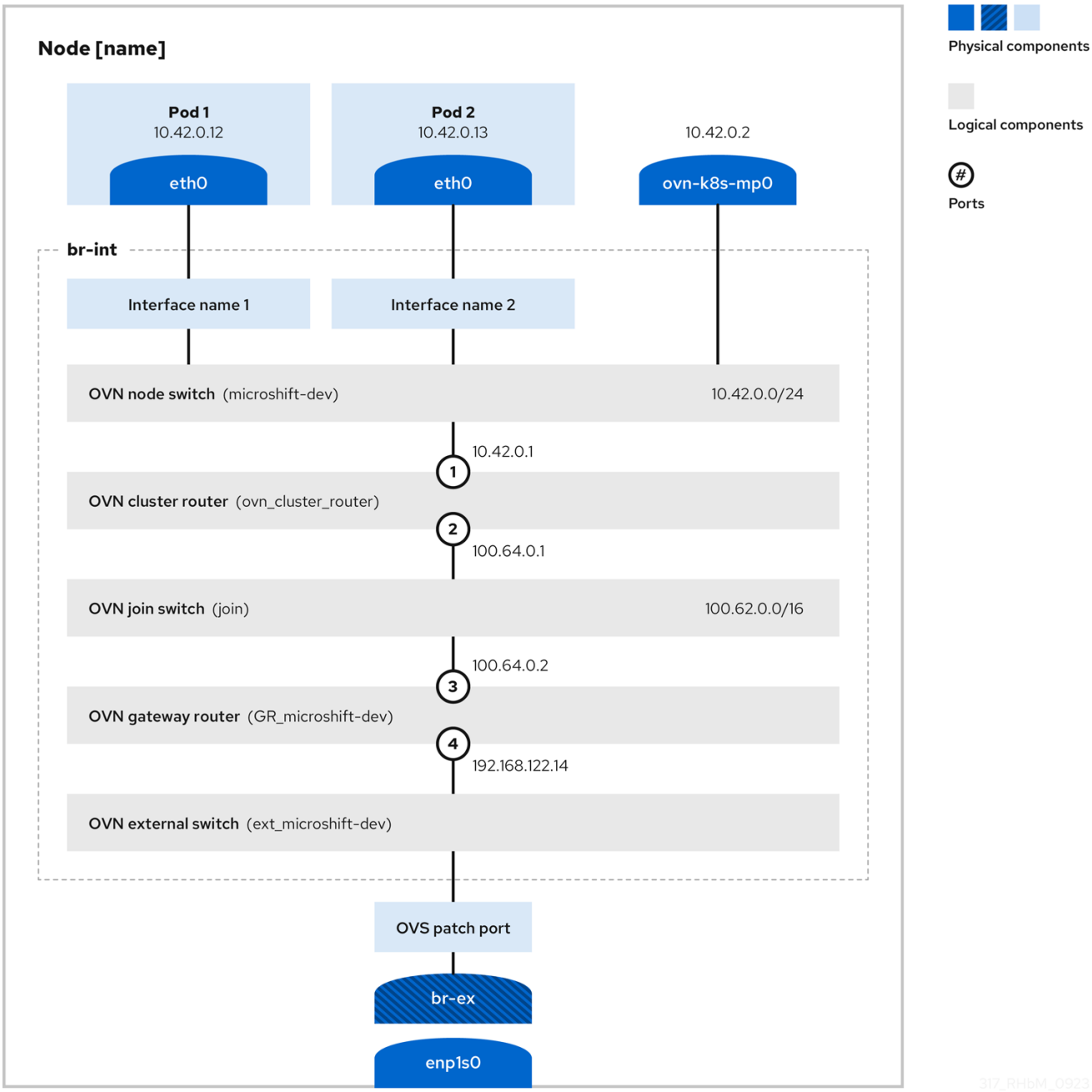
1.7. 网络拓扑

OVN-Kubernetes 提供基于 overlay 的网络实现。此覆盖包括基于 OVS 的服务和 NetworkPolicy 实施。覆盖网络使用 Geneve (Generic Network Virtualization Encapsulation) 隧道协议。如果 Geneve 隧道没有配置，则 Geneve 隧道的 pod 最大传输单元(MTU)被设置为默认路由 MTU。

要配置 MTU，您必须设置等于或小于主机上物理接口的 MTU 的值。MTU 的 less-than 值为传输前添加到隧道头所需的信息留出空间。

OVS 作为 systemd 服务在 MicroShift 节点上运行。OVS RPM 软件包作为对 **microshift-networking** RPM 软件包的依赖项安装。安装了 **microshift-networking** RPM 时，OVS 会立即启动。

红帽构建的 MicroShift 网络拓扑



1.7.1. 虚拟化网络的 OVN 逻辑组件描述

OVN 节点交换机

一个名为 **<node-name>** 的虚拟交换机。OVN 节点交换机根据节点的主机名命名。

- 在本例中，**node-name** 是 **microshift-dev**。

OVN 集群路由器

名为 **ovn_cluster_router** 的虚拟路由器，也称为分布式路由器。

- 在本例中，集群网络是 **10.42.0.0/16**。

OVN join 开关

名为 **join** 的虚拟交换机。

OVN 网关路由器

名为 **GR_<node-name>** 的虚拟路由器，也称为外部网关路由器。

OVN 外部交换机

名为 **ext_<node-name>.** 的虚拟交换机。

1.7.2. 网络拓扑图中的连接描述

- 网络服务和 OVN 外部交换机 **ext_microshift-dev** 之间的南北流量由网关网桥 **br-ex** 通过主机内核提供。
- OVN 网关路由器 **GR_microshift-dev** 通过逻辑路由器端口 4 连接到外部网络交换机 **ext_microshift-dev**。端口 4 附加到节点 IP 地址 192.168.122.14。
- join 交换机 **join** 将 OVN 网关路由器 **GR_microshift-dev** 连接到 OVN 集群路由器 **ovn_cluster_router**。IP 地址范围为 100.62.0.0/16。
    - OVN 网关路由器 **GR_microshift-dev** 通过逻辑路由器端口 3 连接到 OVN **join** 交换机。端口 3 与内部 IP 地址 100.64.0.2 连接。
    - OVN 集群路由器 **ovn_cluster_router** 通过逻辑路由器端口 2 连接到 **join** 交换机。端口 2 与内部 IP 地址 100.64.0.1 连接。
- OVN 集群路由器 **ovn_cluster_router** 通过逻辑路由器端口 1 连接到节点交换机 **microshift-dev**。端口 1 与 OVN 集群网络 IP 地址 10.42.0.1 附加。
- pod 和网络服务之间的东西流量由 OVN 集群路由器 **ovn_cluster_router** 和节点交换机 **microshift-dev** 提供。IP 地址范围为 10.42.0.0/24。
- pod 之间的东西流量由节点交换机 **microshift-dev** 提供，而无需网络地址转换 (NAT)。
- pod 和外部网络之间的南北流量由 OVN 集群路由器 **ovn_cluster_router** 和主机网络提供。此路由器通过 **ovn-kubernetes** 管理端口 **ovn-k8s-mp0** 连接，IP 地址为 10.42.0.2。
- 所有 pod 都通过其接口连接到 OVN 节点交换机。
    - 在本例中，Pod 1 和 Pod 2 通过 **Interface 1** 和 **Interface 2** 连接到节点交换机。

1.8. 其他资源

- 使用 YAML 配置文件
- 了解网络设置
- 关于使用多个网络
- 关于网络策略

第 2 章 了解网络设置

了解如何将网络自定义和默认设置应用到 MicroShift 部署。每个节点都包含在一个机器和单个 MicroShift 中，因此每个部署都需要单独的配置、Pod 和设置。

集群管理员有几个选项用于公开集群内的应用程序到外部流量并确保网络连接：

- 服务，如 NodePort
- API 资源，如 **Ingress** 和 **Route**

默认情况下，Kubernetes 为 pod 内运行的应用分配内部 IP 地址。Pod 及其容器之间可以有网络流量，但集群外的客户端无法直接访问容器集，除非通过一个服务（如 NodePort）公开。

2.1. 创建 OVN-KUBERNETES 配置文件

如果没有创建 OVN-Kubernetes 配置文件，MicroShift 将使用内置默认 OVN-Kubernetes 值。您可以将 OVN-Kubernetes 配置文件写入 **/etc/microshift/ovn.yaml**。为您的配置提供了一个示例文件。

流程

1. 要创建 **ovn.yaml** 文件，请运行以下命令：

```
$ sudo cp /etc/microshift/ovn.yaml.default /etc/microshift/ovn.yaml
```

2. 要列出您创建的配置文件的内容，请运行以下命令：

```
$ cat /etc/microshift/ovn.yaml
```

带有默认最大传输单元(MTU)值的 YAML 文件示例

```
mtu: 1400
```

3. 要自定义配置，您可以更改 MTU 值。以下列表提供了详情：

表 2.1. 支持 MicroShift 的可选 OVN-Kubernetes 配置

字段	类型	Default（默认）	描述	Example
mtu	uint32	auto	用于 pod 的 MTU 值	1300

重要

如果更改了 **ovn.yaml** 文件中的 **mtu** 配置值，您必须重启红帽构建的 MicroShift 运行的主机以应用更新的设置。

自定义 ovn.yaml 配置文件示例

```
mtu: 1300
```

2.2. 重启 OVNKUBE-MASTER POD

以下流程重启 **ovnkube-master** pod。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 使用具有 **cluster-admin** 角色的用户访问集群。
- 在使用 OVN-Kubernetes 网络插件配置的基础架构上安装集群。
- KUBECONFIG 环境变量被设置。

流程

使用以下步骤重启 **ovnkube-master** pod。

1. 运行以下命令来访问远程集群：

```
$ export KUBECONFIG=$PWD/kubeconfig
```

2. 运行以下命令，查找您要重启的 **ovnkube-master** pod 的名称：

```
$ pod=$(oc get pods -n openshift-ovn-kubernetes | awk -F " " '/ovnkube-master/{print $1}')
```

3. 运行以下命令来删除 **ovnkube-master** pod：

```
$ oc -n openshift-ovn-kubernetes delete pod $pod
```

4. 使用以下命令确认新的 **ovnkube-master** pod 正在运行：

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

正在运行的 Pod 列表显示新的 **ovnkube-master** pod 名称和年龄。

2.3. 在 HTTP 或 HTTPS 代理后部署 MICROSHIFT

在您要向 pod 添加基本匿名和安全措施时，在 HTTP 或 HTTPS 代理后部署 MicroShift 集群。

在代理后面部署 MicroShift 时，您必须将主机操作系统配置为使用代理服务以及启动 HTTP 或 HTTPS 请求的所有组件。

所有特定于用户的工作负载或带有出口流量的 pod （如访问云服务）都必须配置为使用代理。MicroShift 中没有内置透明代理的出口流量。

2.4. 使用 RPM-OSTREE HTTP 或 HTTPS 代理

要在 RPM-OStree 中使用 HTTP 或 HTTPS 代理，您必须在配置文件中添加 **Service** 部分，并为 **rpm-ostreed** 服务设置 **http_proxy** 环境变量。

流程

1. 将此设置添加到 **/etc/systemd/system/rpm-ostreed.service.d/00-proxy.conf** 文件中：

```
[Service]
Environment="http_proxy=http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_SERVER:$PROXY_PORT/"
```

2. 接下来，重新加载配置设置并重新启动服务以应用您的更改。

   a. 运行以下命令来重新载入配置设置：

   ```
   $ sudo systemctl daemon-reload
   ```

   b. 运行以下命令重启 **rpm-ostreed** 服务：

   ```
   $ sudo systemctl restart rpm-ostreed.service
   ```

2.5. 在 CRI-O 容器运行时中使用代理

要在 **CRI-O** 中使用 HTTP 或 HTTPS 代理，您必须在配置文件中添加 **Service** 部分并设置 **HTTP_PROXY** 和 **HTTPS_PROXY** 环境变量。您还可以设置 **NO_PROXY** 变量，将主机列表排除在代理之外。

流程

1. 如果配置文件不存在，请为配置文件创建该目录：

   ```
   $ sudo mkdir /etc/systemd/system/crio.service.d/
   ```

2. 在 **/etc/systemd/system/crio.service.d/00-proxy.conf** 文件中添加以下设置：

   ```
   [Service]
   Environment=NO_PROXY="localhost,127.0.0.1"
   Environment=HTTP_PROXY="http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_SERVER:$PROXY_PORT/"
   Environment=HTTPS_PROXY="http://$PROXY_USER:$PROXY_PASSWORD@$PROXY_SERVER:$PROXY_PORT/"
   ```

   重要

   您必须为环境变量定义配置文件的 **Service** 部分，或者代理设置无法应用。

3. 重新载入配置设置：

   ```
   $ sudo systemctl daemon-reload
   ```

4. 重启 CRI-O 服务：

   ```
   $ sudo systemctl restart crio
   ```

5. 重启 MicroShift 服务以应用设置：

   ```
   $ sudo systemctl restart microshift
   ```

验证

1. 运行以下命令检查输出来验证 pod 是否已启动：

```
$ oc get all -A
```

2. 运行以下命令并检查输出，验证 MicroShift 是否可以拉取容器镜像：

```
$ sudo crictl images
```

2.6. 从正在运行的集群获取 OVS 接口的快照

快照代表 OVS 接口在特定时间点的状态和数据。

流程

- 要查看正在运行的 MicroShift 集群中 OVS 接口的快照，请使用以下命令：

```
$ sudo ovs-vsctl show
```

正在运行的集群中的 OVS 接口示例

```
9d9f5ea2-9d9d-4e34-bbd2-dbac154fdc93
    Bridge br-ex
        Port br-ex
            Interface br-ex
                type: internal
        Port patch-br-ex_localhost.localdomain-to-br-int 1
            Interface patch-br-ex_localhost.localdomain-to-br-int
                type: patch
                options: {peer=patch-br-int-to-br-ex_localhost.localdomain} 2
    Bridge br-int
        fail_mode: secure
        datapath_type: system
        Port patch-br-int-to-br-ex_localhost.localdomain
            Interface patch-br-int-to-br-ex_localhost.localdomain
                type: patch
                options: {peer=patch-br-ex_localhost.localdomain-to-br-int}
        Port eebee1ce5568761
            Interface eebee1ce5568761 3
        Port b47b1995ada84f4
            Interface b47b1995ada84f4 4
        Port "3031f43d67c167f"
            Interface "3031f43d67c167f" 5
        Port br-int
            Interface br-int
                type: internal
        Port ovn-k8s-mp0 6
            Interface ovn-k8s-mp0
                type: internal
    ovs_version: "2.17.3"
```

1. **patch-br-ex_localhost.localdomain-to-br-int** 和 **patch-br-int-to-br-ex_localhost.localdomain** 是连接 **br-ex** 和 **br-int** 的 OVS 补丁端口。

2 **patch-br-ex_localhost.localdomain-to-br-int** 和 **patch-br-int-to-br-ex_localhost.localdomain** 是连接 **br-ex** 和 **br-int** 的 OVS 补丁端口。

3 pod 接口 **eebee1ce5568761** 使用 pod 沙盒 ID 的前 15 位命名，并插入到 **br-int** 网桥。

4 pod 接口 **b47b1995ada84f4** 使用 pod 沙盒 ID 的前 15 位命名，并插入到 **br-int** 网桥中。

5 pod 接口 **3031f43d67c167f** 使用 pod 沙盒 ID 的前 15 位命名，并插入到 **br-int** 网桥中。

6 hairpin 流量的 OVS 内部端口，**ovn-k8s-mp0** 由 **ovnkube-master** 容器创建。

2.7. 用于工作负载的 MICROSHIFT LOADBALANCER 服务

MicroShift 具有网络负载均衡器的内置实现，可用于集群中的工作负载和应用程序。您可以通过将 pod 配置为解释入口规则并充当入口控制器来创建 **LoadBalancer** 服务。以下流程提供了基于部署的 **LoadBalancer** 服务的示例。

2.8. 为应用程序部署负载均衡器

以下示例使用节点 IP 地址作为 **LoadBalancer** 服务配置文件的外部 IP 地址。使用本示例作为如何部署负载均衡器的指导。

先决条件

- 已安装 OpenShift CLI (**oc**)。
- 在使用 OVN-Kubernetes 网络插件配置的基础架构上安装集群。
- **KUBECONFIG** 环境变量被设置。

流程

1. 输入以下命令验证您的 pod 是否正在运行：

```
$ oc get pods -A
```

输出示例

```
NAMESPACE                  NAME                                                READY   STATUS
RESTARTS  AGE
default                    i-06166fbb376f14a8bus-west-2computeinternal-debug-qtwcr  1/1
Running   0     46m
kube-system                csi-snapshot-controller-5c6586d546-lprv4            1/1
Running   0     51m
kube-system                csi-snapshot-webhook-6bf8ddc7f5-kz6k9               1/1
Running   0     51m
openshift-dns              dns-default-45jl7                                   2/2     Running   0
50m
openshift-dns              node-resolver-7wmzf                                 1/1     Running   0
51m
openshift-ingress          router-default-78b86fbf9d-qvj9s                     1/1     Running
0     51m
openshift-multus           dhcp-daemon-j7qnf                                   1/1     Running   0
```

```
51m
openshift-multus                multus-r758z                                  1/1     Running    0
51m
openshift-operator-lifecycle-manager catalog-operator-85fb86fcb9-t6zm7                1/1
Running    0     51m
openshift-operator-lifecycle-manager olm-operator-87656d995-fvz84                     1/1
Running    0     51m
openshift-ovn-kubernetes            ovnkube-master-5rfhh                          4/4     Running
0     51m
openshift-ovn-kubernetes            ovnkube-node-gcnt6                            1/1     Running
0     51m
openshift-service-ca                service-ca-bf5b7c9f8-pn6rk                    1/1     Running
0     51m
openshift-storage                   topolvm-controller-549f7fbdd5-7vrmv                5/5
Running    0     51m
openshift-storage                   topolvm-node-rht2m                            3/3     Running    0
50m
```

2. 运行以下命令来创建命名空间：

```
$ NAMESPACE=<nginx-lb-test> 1
```

1 将 _<nginx-lb-test> 替换为您要创建的应用程序命名空间。

```
$ oc create ns $NAMESPACE
```

命名空间示例

以下示例在创建的命名空间中部署测试 **nginx** 应用程序的三个副本：

```
oc apply -n $NAMESPACE -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx
data:
  headers.conf: |
    add_header X-Server-IP  \$server_addr always;
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

```
      - image: quay.io/packit/nginx-unprivileged
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: nginx-configs
          subPath: headers.conf
          mountPath: /etc/nginx/conf.d/headers.conf
        securityContext:
          allowPrivilegeEscalation: false
          seccompProfile:
            type: RuntimeDefault
          capabilities:
            drop: ["ALL"]
          runAsNonRoot: true
      volumes:
        - name: nginx-configs
          configMap:
            name: nginx
            items:
              - key: headers.conf
                path: headers.conf
EOF
```

3. 您可以运行以下命令来验证三个副本是否已成功启动：

```
$ oc get pods -n $NAMESPACE
```

4. 运行以下命令，为 **nginx** 测试应用程序创建 **LoadBalancer** 服务：

```
oc create -n $NAMESPACE -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  ports:
  - port: 81
    targetPort: 8080
  selector:
    app: nginx
  type: LoadBalancer
EOF
```

注意

您必须确保 **port** 参数是一个没有被其他 **LoadBalancer** 服务或 MicroShift 组件占用的主机端口。

5. 运行以下命令，验证服务文件是否存在，是否正确分配了外部 IP 地址，并且外部 IP 与节点 IP 相同：

```
$ oc get svc -n $NAMESPACE
```

输出示例

```
NAME    TYPE          CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
nginx   LoadBalancer  10.43.183.104   192.168.1.241   81:32434/TCP   2m
```

验证

以下命令使用 **LoadBalancer** 服务配置的外部 IP 地址形成五个到示例 **nginx** 应用程序的连接。命令的结果是这些服务器 IP 地址的列表。

- 运行以下命令，验证负载均衡器是否向所有正在运行的应用程序发送请求：

```
EXTERNAL_IP=192.168.1.241
seq 5 | xargs -Iz curl -s -I http://$EXTERNAL_IP:81 | grep X-Server-IP
```

如果 **LoadBalancer** 服务成功将流量分发到应用程序，则上一命令的输出包含不同的 IP 地址，例如：

输出示例

```
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.43
X-Server-IP: 10.42.0.41
X-Server-IP: 10.42.0.43
```

2.9. 阻止对特定主机接口上 NODEPORT 服务的外部访问

OVN-Kubernetes 不限制可以从红帽构建的 MicroShift 节点外部访问 NodePort 服务的主机接口。以下流程解释了如何在特定主机接口上阻止 NodePort 服务并限制外部访问。

先决条件

- 您必须具有具有 root 特权的帐户。

流程

1. 运行以下命令，将 **NODEPORT** 变量更改为分配给 Kubernetes NodePort 服务的主机端口号：

```
# export NODEPORT=30700
```

2. 将 **INTERFACE_IP** 值从您要阻止的主机接口更改为 IP 地址。例如：

```
# export INTERFACE_IP=192.168.150.33
```

3. 在 **nat** 表 PREROUTING 链中插入一条新规则，以丢弃与目标端口和 IP 地址匹配的所有数据包。例如：

```
$ sudo nft -a insert rule ip nat PREROUTING tcp dport $NODEPORT ip daddr
$INTERFACE_IP drop
```

4. 运行以下命令列出新规则：

```
$ sudo nft -a list chain ip nat PREROUTING
table ip nat {
 chain PREROUTING { # handle 1
  type nat hook prerouting priority dstnat; policy accept;
  tcp dport 30700 ip daddr 192.168.150.33 drop # handle 134
  counter packets 108 bytes 18074 jump OVN-KUBE-ETP # handle 116
  counter packets 108 bytes 18074 jump OVN-KUBE-EXTERNALIP # handle 114
  counter packets 108 bytes 18074 jump OVN-KUBE-NODEPORT # handle 112
 }
}
```

注意

请记录下新添加的规则的 **handle** 号。您需要删除以下步骤中的 **handle** 号。

5. 使用以下示例命令删除自定义规则：

```
$ sudo nft -a delete rule ip nat PREROUTING handle 134
```

2.10. 多播 DNS 协议

多播 DNS 协议 (mDNS) 允许使用在 **5353/UDP** 端口上公开的多播进行名称解析和服务发现。

MicroShift 包括一个嵌入式 mDNS 服务器用于部署场景，在这种情况下，无法重新配置权威 DNS 服务器来将客户端指向 MicroShift 上的服务。嵌入式 DNS 服务器允许 MicroShift 公开的 **.local** 域由 LAN 上的其他元素发现。

2.11. 审计公开的网络端口

在 MicroShift 上，可以在以下情况下由工作负载打开主机端口。您可以检查日志以查看网络服务。

2.11.1. hostNetwork

当使用 **hostNetwork:true** 设置配置 pod 时，pod 在主机网络命名空间中运行。此配置可以独立打开主机端口。MicroShift 组件日志无法用于跟踪此问题单，端口会受到 firewalld 规则的约束。如果端口在 firewalld 中打开，您可以查看 firewalld 调试日志中打开的端口。

先决条件

- 有访问构建主机的 root 用户。

流程

1. 可选：您可以使用以下示例命令检查 ovnkube-node pod 中是否设置了 **hostNetwork:true** 参数：

```
$ sudo oc get pod -n openshift-ovn-kubernetes <ovnkube-node-pod-name> -o json | jq -r
'.spec.hostNetwork' true
```

2. 运行以下命令，在 firewalld 日志中启用 debug：

```
$ sudo vi /etc/sysconfig/firewalld
FIREWALLD_ARGS=--debug=10
```

3. 重启 firewalld 服务：

```
$ sudo systemctl restart firewalld.service
```

4. 要验证 debug 选项是否已正确添加，请运行以下命令：

```
$ sudo systemd-cgls -u firewalld.service
```

firewalld 调试日志存储在 **/var/log/firewalld** 路径中。

添加端口打开规则时的日志示例：

```
2023-06-28 10:46:37 DEBUG1: config.getZoneByName('public')
2023-06-28 10:46:37 DEBUG1: config.zone.7.addPort('8080', 'tcp')
2023-06-28 10:46:37 DEBUG1: config.zone.7.getSettings()
2023-06-28 10:46:37 DEBUG1: config.zone.7.update('...')
2023-06-28 10:46:37 DEBUG1: config.zone.7.Updated('public')
```

当删除端口打开规则时的日志示例：

```
2023-06-28 10:47:57 DEBUG1: config.getZoneByName('public')
2023-06-28 10:47:57 DEBUG2: config.zone.7.Introspect()
2023-06-28 10:47:57 DEBUG1: config.zone.7.removePort('8080', 'tcp')
2023-06-28 10:47:57 DEBUG1: config.zone.7.getSettings()
2023-06-28 10:47:57 DEBUG1: config.zone.7.update('...')
2023-06-28 10:47:57 DEBUG1: config.zone.7.Updated('public')
```

2.11.2. hostPort

您可以在 MicroShift 中访问 hostPort 设置日志。以下日志是 hostPort 设置的示例：

流程

- 您可以运行以下命令来访问日志：

```
$ journalctl -u crio | grep "local port"
```

打开主机端口时 CRI-O 日志示例：

```
$ Jun 25 16:27:37 rhel92 crio[77216]: time="2023-06-25 16:27:37.033003098+08:00"
level=info msg="Opened local port tcp:443"
```

主机端口关闭时的 CRI-O 日志示例：

```
$ Jun 25 16:24:11 rhel92 crio[77216]: time="2023-06-25 16:24:11.342088450+08:00"
level=info msg="Closing host port tcp:443"
```

2.11.3. NodePort 和 LoadBalancer 服务

OVN-Kubernetes 为 **NodePort** 和 **LoadBalancer** 服务类型打开主机端口。这些服务添加 iptables 规则，该规则使用来自主机端口的入口流量并将其转发到 clusterIP。以下示例中显示了 **NodePort** 和 **LoadBalancer** 服务的日志：

流程

1. 要访问 **ovnkube-master** pod 的名称，请运行以下命令：

```
$ oc get pods -n openshift-ovn-kubernetes | awk '/ovnkube-master/{print $1}'
```

ovnkube-master pod 名称示例

```
ovnkube-master-n2shv
```

2. 您可以使用 **ovnkube-master** pod 访问 **NodePort** 和 **LoadBalancer** 服务日志，并运行以下命令：

```
$ oc logs -n openshift-ovn-kubernetes <ovnkube-master-pod-name> ovnkube-master | grep -E "OVN-KUBE-NODEPORT|OVN-KUBE-EXTERNALIP"
```

NodePort 服务：

当主机端口打开时，ovnkube-master pod 的 ovnkube-master 容器中的日志示例：

```
$ I0625 09:07:00.992980 2118395 iptables.go:27] Adding rule in table: nat, chain: OVN-KUBE-NODEPORT with args: "-p TCP -m addrtype --dst-type LOCAL --dport 32718 -j DNAT --to-destination 10.96.178.142:8081" for protocol: 0
```

当主机端口关闭时，ovnkube-master pod 的 ovnkube-master 容器中的日志示例：

```
$ Deleting rule in table: nat, chain: OVN-KUBE-NODEPORT with args: "-p TCP -m addrtype --dst-type LOCAL --dport 32718 -j DNAT --to-destination 10.96.178.142:8081" for protocol: 0
```

LoadBalancer 服务：

当主机端口打开时，ovnkube-master pod 的 ovnkube-master 容器中的日志示例：

```
$ I0625 09:34:10.406067  128902 iptables.go:27] Adding rule in table: nat, chain: OVN-KUBE-EXTERNALIP with args: "-p TCP -d 172.16.47.129 --dport 8081 -j DNAT --to-destination 10.43.114.94:8081" for protocol: 0
```

当主机端口关闭时，ovnkube-master pod 的 ovnkube-master 容器中的日志示例：

```
$ I0625 09:37:00.976953  128902 iptables.go:63] Deleting rule in table: nat, chain: OVN-KUBE-EXTERNALIP with args: "-p TCP -d 172.16.47.129 --dport 8081 -j DNAT --to-destination 10.43.114.94:8081" for protocol: 0
```

第 3 章 了解并配置路由器

了解使用 MicroShift 配置路由器和路由准入策略的默认和自定义设置。

3.1. 关于配置路由器

要使入口可选，您可以配置 MicroShift 入口路由器设置来管理哪些端口（若有）会公开给网络流量。指定的路由是入口负载均衡的示例。

- 默认入口路由器始终在 **http: 80** 和 **https: 443** 端口上的所有 IP 地址上运行。
- 默认路由器设置允许访问任何命名空间。

在 MicroShift 上运行的一些应用程序可能不需要默认路由器，而是创建自己的。您可以将路由器配置为控制入口和命名空间访问。

提示

在使用 **oc get deployment -n openshift-ingress** 命令开始配置前，您可以检查 MicroShift 安装中的默认路由器是否存在，该命令返回以下输出：

```
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
router-default   1/1     1            1           2d23h
```

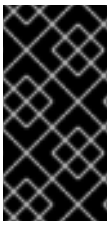
3.1.1. 路由器设置和有效值

入口路由器设置由以下参数和有效值组成：

config.yaml 路由器设置示例

```
# ...
ingress:
  listenAddress:
    - "" (1)
  ports: (2)
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed (3)
  status: Managed (4)
# ...
```

1. **ingress.listenAddress** 值默认为主机的整个网络。有效的自定义值可以是单个 IP 地址或主机名，也可以是 IP 地址或主机名列表。
2. 两个端口条目的有效值为 1-65535 范围内的单个唯一端口。**ports.http** 和 **ports.https** 字段的值不能相同。
3. 默认值。允许路由在命名空间间声明同一主机名的不同路径。
4. 默认值。入口端口需要管理才能保持打开状态。



重要

firewalld 服务由默认的 MicroShift 路由器和启用路由器的配置绕过。在路由器激活时，必须通过设置网络策略来控制入口和出口。

3.2. 禁用路由器

在工业 IoT 空间等用例中，MicroShift pod 只需要连接到南向操作系统和北向云数据系统，则不需要入站服务。使用这个流程在这样的仅出口用例中禁用路由器。

先决条件

- 已安装 MicroShift。
- 您创建了 MicroShift config.yaml 文件。
- 已安装 OpenShift CLI (oc)。

提示

如果同时完成 MicroShift config.yaml 文件中需要进行的所有配置，您可以最小化系统重启。

流程

1. 在 MicroShift config.yaml 文件中将 ingress.status 字段的值更新为 Removed，如下例所示：

config.yaml ingress 小节示例

```
# ...
ingress:
  ports:
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed
  status: Removed (1)
# ...
```

-

1

当值设为 Removed 时，ingress.ports 中列出的端口会自动关闭。ingress 小节中的任何其他设置都会被忽略，例如 routeAdmissionPolicy.namespaceOwnership 字段中的任何值。

2. 运行以下命令来重启 MicroShift 服务：

```
$ sudo systemctl restart microshift
```

注意

MicroShift 服务会在重启过程中输出当前的配置。

验证

- 系统重启后，运行以下命令来验证路由器是否已移除，并且是否停止了入口：

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

预期输出

```
No resources found in openshift-ingress namespace.
```

3.3. 配置路由器入口

如果您的 MicroShift 应用程序只需要侦听数据流量，您可以配置 listenAddress 设置来隔离您的设备。您还可以为网络连接配置特定的端口和 IP 地址。使用所需的组合来为您的用例自定义端点配置。

3.3.1. 配置路由器端口

您可以通过配置路由器入口字段来控制设备使用哪些端口。

先决条件

- 已安装 MicroShift。
- 您创建了 MicroShift config.yaml 文件。
- 已安装 OpenShift CLI (oc)。

提示

如果同时完成 MicroShift config.yaml 文件中需要进行的所有配置，您可以最小化系统重启。

流程

1. 将 ingress.ports.http 和 ingress.ports.https 字段中的 MicroShift config.yaml 端口值更新至您要使用的端口：

config.yaml 路由器设置示例

```
# ...
ingress:
  ports: 1
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed
  status: Managed 2
# ...
```

1 显示的默认端口。可自定义.两个端口条目的有效值为 1-65535 范围内的单个唯一端口。ports.http 和 ports.https 字段的值不能相同。

2 默认值。入口端口需要管理才能保持打开状态。

2. 运行以下命令来重启 MicroShift 服务：

```
$ sudo systemctl restart microshift
```

3.3.2. 配置路由器 IP 地址

您可以通过配置特定的 IP 地址，将网络流量限制到路由器。例如：

- 用例，路由器只能在内部网络上访问，但不能在北向公共网络上访问
- 用例，只有通过北向公共网络访问路由器，但不适用于内部网络
- 例如，路由器可以被内部网络和北向公共网络访问，但在单独的 IP 地址上

先决条件

- 已安装 MicroShift。
- 您创建了 MicroShift config.yaml 文件。
- 已安装 OpenShift CLI (oc)。

提示

如果同时完成 MicroShift config.yaml 文件中需要进行的所有配置，您可以最小化系统重启。

流程

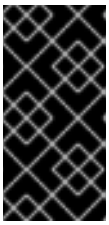
1. 根据您的要求更新 MicroShift config.yaml 中的 ingress.listenAddress 字段中的列表，如下例所示：

默认路由器 IP 地址列表

```
# ...
ingress:
  listenAddress:
    - "<host_network>" 1
# ...
```

1

ingress.listenAddress 值默认为主机的整个网络。要继续使用默认列表，请从 MicroShift config.yaml 文件中删除 listen.Address 字段。要自定义此参数，请使用 list。列表中可以包含单个 IP 地址或 NIC 名称，或者多个 IP 地址和 NIC 名称。



重要

您必须删除 listenAddress 参数，或使用 config.yaml 文件以列表的形式添加值。不要将字段留空，或者 MicroShift 在重启后崩溃。

带有单一主机 IP 地址的路由器设置示例

```
# ...
ingress:
  listenAddress:
    - 10.2.1.100
# ...
```

带有 IP 地址和 NIC 名称的路由器设置示例

```
# ...
ingress:
  listenAddress:
```

```
    - 10.2.1.100
    - 10.2.2.10
    - ens3
# ...
```

2. 运行以下命令来重启 MicroShift 服务：

```
$ sudo systemctl restart microshift
```

验证

- 要验证是否应用了您的设置，请确保 ingress.listenAddress IP 地址可以访问，然后您可以使用目的地向其中一个负载均衡器 IP 地址 curl 路由。

3.4. 其他资源

- 默认设置 (MicroShift)
- 关于网络策略

3.5. 配置路由准入策略

默认情况下，MicroShift 允许多个命名空间中的路由使用相同的主机名。您可以通过配置路由准入策略来防止路由在不同命名空间中声明相同的主机名。

先决条件

- 已安装 MicroShift。
- 您创建了 MicroShift config.yaml 文件。
- 已安装 OpenShift CLI（oc）。

提示

如果同时完成 MicroShift config.yaml 文件中需要进行的所有配置，您可以最小化系统重启。

流程

1. 要防止不同命名空间中的路由声明同一主机名，请在 MicroShift config.yaml 文件中将 namespaceOwnership 字段值更新为 Strict。请参见以下示例：

config.yaml 路由准入策略示例

```
# ...
ingress:
  routeAdmissionPolicy:
    namespaceOwnership: Strict (1)
# ...
```

(1) 防止不同命名空间中的路由声明同一主机。有效值为 Strict 和 InterNamespaceAllowed。如果您删除自定义 config.yaml 中的值，则会自动设置 InterNamespaceAllowed 值。

2. 要应用配置，请运行以下命令重启 MicroShift 服务：

```
$ sudo systemctl restart microshift
```

第 4 章 网络策略

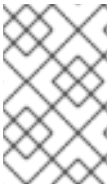
4.1. 关于网络策略

了解网络策略对 MicroShift 的工作原理，以限制或允许到集群中的 pod 的网络流量。

4.1.1. 网络策略在 MicroShift 中如何工作

在使用 MicroShift 的默认 OVN-Kubernetes Container Network Interface (CNI)插件的集群中，网络隔离由 firewalld 控制，它在主机上配置，并由 MicroShift 中创建的 NetworkPolicy 对象。支持同时使用 firewalld 和 NetworkPolicy。

- 网络策略仅在 OVN-Kubernetes 控制的流量范围内工作，以便它们可以应用到除启用了 hostPort/hostNetwork 的 pod 以外的每个情况。
- firewalld 设置也适用于启用了 hostPort/hostNetwork 的 pod。



注意

在强制任何 NetworkPolicy 之前运行 firewalld 规则。

警告

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

网络策略无法阻止来自 localhost 的流量。

默认情况下，MicroShift 节点中的所有 pod 都可从其他 pod 和网络端点访问。要隔离集群中的一个或多个 pod，您可以创建 NetworkPolicy 对象来指示允许的入站连接。您可以创建和删除 NetworkPolicy 对象。

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

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

以下示例 NetworkPolicy 对象演示了支持不同的情景：

- 拒绝所有流量：

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

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

- 允许来自默认路由器的连接，这是 MicroShift 中的入口：

  要允许 MicroShift 默认路由器的连接，请添加以下 NetworkPolicy 对象：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  podSelector: {}
  policyTypes:
  - Ingress
```

- 仅接受同一命名空间中的 pod 的连接：

  要使 pod 接受同一命名空间中其他 pod 的连接，但拒绝其他命名空间中所有 pod 的连接，请添加以下 NetworkPolicy 对象：

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

- 仅允许基于 pod 标签的 HTTP 和 HTTPS 流量：

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

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

- 使用命名空间和 pod 选择器接受连接：

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

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

```
      project: project_name
  podSelector:
    matchLabels:
      name: test-pods
```

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

例如，对于上例中定义的 NetworkPolicy 对象，您可以定义 allow-same-namespace 和 allow-http-and-https 策略。该配置允许带有标签 role=frontend 的 pod 接受每个策略允许的任何连接。即，任何端口上来自同一命名空间中的 pod 的连接，以及端口 80 和 443 上的来自任意命名空间中 pod 的连接。

4.1.2. 使用 OVN-Kubernetes 网络插件优化网络策略

在设计您的网络策略时，请参考以下指南：

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

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

  例如，以下策略包含两个规则：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

以下策略表示这两个规则与以下相同的规则：

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

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

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

以下网络策略将这两个相同的规则作为一个：

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

```
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

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

4.2. 创建网络策略

您可以为命名空间创建网络策略。

4.2.1. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

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

(1) NetworkPolicy 对象的名称。

(2)

描述策略应用到的 pod 的选择器。

3

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

4

接受流量的一个或多个目标端口的列表。

4.2.2. 使用 CLI 创建网络策略

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

先决条件

- 已安装 OpenShift CLI（oc）。
- 您在网络策略要应用到的命名空间中。

流程

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

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

其中：

<policy_name>

指定网络策略文件名。

b. 在您刚才创建的文件中定义网络策略，如下例所示：

拒绝来自所有命名空间中的所有 pod 的入口流量

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

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

允许来自所有命名空间中的所有 pod 的入口流量

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

允许从特定命名空间中到一个 pod 的入口流量

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-traffic-pod
spec:
  podSelector:
   matchLabels:
     pod: pod-a
```

```
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

2. 运行以下命令来创建网络策略对象：

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

其中：

<policy_name>

指定网络策略文件名。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

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

4.2.3. 创建默认拒绝所有网络策略

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

先决条件

- 已安装 OpenShift CLI（oc）。

- 您在网络策略要应用到的命名空间中。

流程

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

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

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

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

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

2. 输入以下命令应用策略：

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

输出示例

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

4.2.4. 创建网络策略以允许来自外部客户端的流量

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

注意

在强制任何 NetworkPolicy 之前运行 firewalld 规则。

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

- 已安装 OpenShift CLI（oc）。
- 您在网络策略要应用到的命名空间中。

流程

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

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

2. 输入以下命令应用策略：

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

输出示例

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

4.2.5. 创建网络策略，允许从所有命名空间中到应用程序的流量

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

- 已安装 OpenShift CLI（oc）。
- 您在网络策略要应用到的命名空间中。

流程

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-namespaces
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {} 2
```

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

2 选择所有命名空间中的所有 pod。

注意

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

2. 输入以下命令应用策略：

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

输出示例

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

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

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

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

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

3. 在 shell 中运行以下命令，并观察是否允许请求：

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

预期输出

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

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

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

4.2.6. 创建网络策略，允许从一个命名空间中到应用程序的流量

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

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

先决条件

- 已安装 OpenShift CLI（oc）。
- 您在网络策略要应用到的命名空间中。

流程

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production (2)
```

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

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

2. 输入以下命令应用策略：

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

输出示例

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

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

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

2. 运行以下命令来创建 prod 命名空间：

```
$ oc create namespace prod
```

3. 运行以下命令来标记 prod 命名空间：

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

4. 运行以下命令来创建 dev 命名空间：

```
$ oc create namespace dev
```

5. 运行以下命令来标记 dev 命名空间：

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

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

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

7. 在 shell 中运行以下命令，并观察请求是否被阻止：

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

预期输出

```
wget: download timed out
```

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

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

9. 在 shell 中运行以下命令，并观察是否允许请求：

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

预期输出

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

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

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

4.3. 编辑网络策略

您可以编辑命名空间的现有网络策略。典型的编辑可能包括对策略应用到的 pod 的更改、允许的入口流量以及接受流量的目的地端口。在编辑 NetworkPolicy 对象时，不能更改 apiVersion、kind 和 name 字段，因为这些定义资源本身。

4.3.1. 编辑网络策略

您可以编辑命名空间中的网络策略。

先决条件

- 已安装 OpenShift CLI（oc）。
- 您在网络策略所在的命名空间中。

流程

1. 可选： 要列出一个命名空间中的网络策略对象，请输入以下命令：

```
$ oc get networkpolicy
```

其中：

<namespace>

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

2. 编辑网络策略对象。

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

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

其中：

<namespace>

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

<policy_file>

指定包含网络策略的文件的名称。

- 如果您需要直接更新网络策略对象，请输入以下命令：

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

其中：

<policy_name>

指定网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

3. 确认网络策略对象已更新。

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

其中：

<policy_name>

指定网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

4.3.2. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
```

```
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1 NetworkPolicy 对象的名称。

2 描述策略应用到的 pod 的选择器。

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

4 接受流量的一个或多个目标端口的列表。

4.4. 删除网络策略

您可以从命名空间中删除网络策略。

4.4.1. 使用 CLI 删除网络策略

您可以删除命名空间中的网络策略。

先决条件

- 已安装 OpenShift CLI（oc）。

- 您在网络策略所在的命名空间中。

流程

- 要删除网络策略对象，请输入以下命令：

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

其中：

<policy_name>

指定网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

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

4.5. 查看网络策略

使用以下步骤查看命名空间的网络策略。

4.5.1. 使用 CLI 查看网络策略

您可以检查命名空间中的网络策略。

先决条件

- 已安装 OpenShift CLI（oc）。

- 您在网络策略所在的命名空间中。

流程

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

```
$ oc get networkpolicy
```

  - 可选：要检查特定的网络策略，请输入以下命令：

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

其中：

<policy_name>

指定要检查的网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

例如：

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

oc describe 命令的输出

```
Name:         allow-same-namespace
Namespace:    ns1
Created on:   2021-05-24 22:28:56 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
```

```
  PodSelector:     <none> (Allowing the specific traffic to all pods in this
namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: <none>
  Not affecting egress traffic
  Policy Types: Ingress
```

第 5 章 多网络

5.1. 关于使用多个网络

除了默认的 OVN-Kubernetes Container Network Interface (CNI)插件外，MicroShift Multus CNI 还可用于串联其他 CNI 插件。安装和使用 MicroShift Multus 是可选的。

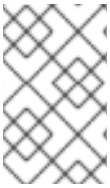
5.1.1. MicroShift 中的额外网络

在集群安装过程中，*默认* pod 网络被配置为使用默认值，除非您自定义配置。默认网络处理集群中的所有一般网络流量。使用 MicroShift Multus CNI 插件，您可以从其他网络向 pod 添加额外的接口。这可让您灵活地配置提供交换或路由等网络功能的 pod。

5.1.1.1. 支持用于网络隔离的额外网络

MicroShift 4.16 支持以下额外网络：

- bridge ：允许同一主机上的 pod 相互通信，并与主机通信。
- IPVLAN：允许主机上的 pod 与其他主机进行通信。
  - 这类似于基于 MACVLAN 的额外网络。
  - 每个 pod 共享与父物理网络接口相同的 MAC 地址，这与基于 MACVLAN 的额外网络不同。
- MACVLAN ：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其他主机上的 pod 通信。附加到基于 MACVLAN 的额外网络的每个 pod 都有唯一的 MAC 地址。



注意

不支持为额外网络设置网络策略。

5.1.1.2. 使用案例：用于网络隔离的其他网络

当需要网络隔离时，您可以使用额外网络，包括 control plane 和数据平面分离。例如，如果您希望 pod 访问主机上的网络，并且也与部署到边缘的设备进行通信，则可以配置额外的接口。这些边缘设备可能位于隔离的 operator 网络中，或者会定期断开连接。

隔离网络流量对以下性能和安全性原因很有用：

性能

您可以在两个不同的平面上发送流量，以管理每个平面上的流量数量。

安全性

您可以将敏感的流量发送到专为安全考虑而管理的网络平面，也可隔离不能在租户或客户间共享的私密数据。

重要

当 MicroShift 服务启动时，Multus CNI 插件会被部署。因此，如果在 MicroShift 启动后添加了 microshift-multus RPM 软件包，则需要主机重启。重启可确保使用 Multus 注解重新创建所有容器。

5.1.1.3. 如何实施额外网络

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 get pod <pod_name> -o=jsonpath='{ .metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status }' 命令来查看 pod 的接口。
- 如果您添加了使用 MicroShift Multus CNI 的额外网络接口，它们名为 net1,net2, …, netN。
- MicroShift Multus DaemonSet 启动时会创建 CNI 配置。此配置是自动生成的，包括默认委托的主要 CNI。对于 MicroShift，默认 CNI 是 OVN-Kubernetes。

5.1.1.4. 如何将额外网络附加到 pod

要将额外网络接口附加到 pod，您必须创建并应用配置来定义接口的附加方式。

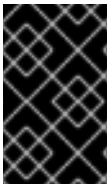
- 您必须配置要使用的任何额外网络。由于网络的不同，不提供默认配置。
- 您必须应用 YAML 清单，以使用 NetworkAttachmentDefinition 自定义资源(CR)指定每个接口。每个 CR 中的配置定义如何创建该接口。
- CRI-O 必须配置为使用 Multus。microshift-multus RPM 中包含了一个默认配置。
  - 如果在现有 MicroShift 实例上安装了 Multus CNI，则必须重启主机。
  - 如果 Multus CNI 与 MicroShift 一起安装，您可以添加 CR 和 pod，然后启动 MicroShift 服务。在这种情况下，不需要重启主机。

5.1.1.5. 额外网络类型的配置

以下部分介绍了额外网络的具体配置字段。

5.1.2. 在正在运行的集群中安装 Multus CNI 插件

如果要将额外网络附加到 pod 以进行高性能网络配置，您可以安装 MicroShift Multus RPM 软件包。安装后，需要重启主机来重新创建带有 Multus 注解的所有 pod。



重要

不支持卸载 Multus CNI 插件。

先决条件

1. 有对主机的 root 访问权限。

流程

1. 运行以下命令来安装 Multus RPM 软件包：

```
$ sudo dnf install microshift-multus
```

提示

如果现在为额外网络创建自定义资源(CR)，您可以使用一个重启完成安装并应用配置。

2. 要将软件包清单应用到活跃集群，请运行以下命令重启主机：

```
$ sudo systemctl restart
```

验证

1. 重启后，运行以下命令来确保创建 Multus CNI 插件组件：

```
$ oc get pod -A | grep multus
```

输出示例

```
openshift-multus     dhcp-daemon-ktzqf     1/1   Running   0     45h
openshift-multus     multus-4frf4          1/1   Running   0     45h
```

后续步骤

1. 如果您还没有这样做，请配置并应用您要使用的额外网络。

2. 部署使用创建的 CR 的应用程序。

5.1.3. 配置桥接额外网络

以下对象描述了 Bridge CNI 插件的配置参数：

表 5.1. bridge CNI 插件 JSON 配置对象

字段	类型	描述
cniVersion	**string**	CNI 规格版本。需要 **0.4.0** 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**。
mtu	**string**	可选：将最大传输单元 (MTU) 设置为指定的值。默认值由内核自动设置。
enabledad	**布尔值**	可选：为容器侧 **veth** 启用重复的地址检测。默认值为 **false**。
macspoofchk	**布尔值**	可选：启用 mac spoof 检查，将来自容器的流量限制为接口的 mac 地址。默认值为 **false**。

5.1.3.1. bridge CNI 插件配置示例

以下示例配置了名为 bridge-conf 的额外网络，以用于 MicroShift Multus CNI：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
```

```
  name: bridge-conf
spec:
  config: '{
      "cniVersion": "0.4.0",
      "type": "bridge",
      "bridge": "test-bridge",
      "mode": "bridge",
      "ipam": {
        "type": "host-local",
        "ranges": [
          [
            {
              "subnet": "10.10.0.0/16",
              "rangeStart": "10.10.1.20",
              "rangeEnd": "10.10.3.50",
              "gateway": "10.10.0.254"
            }
          ]
        ],
        "dataDir": "/var/lib/cni/test-bridge"
      }
    }'
```

5.1.4. 配置 ipvlan 额外网络

以下对象描述了 IPVLAN CNI 插件的配置参数：

表 5.2. 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。

字段	类型	描述
linkInContainer	布尔值	可选：指定 master 接口是否在容器网络命名空间或主网络命名空间中。将值设为 **true** 以请求使用容器命名空间 master 接口。

重要

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

5.1.4.1. IPVLAN CNI 插件配置示例

以下示例配置了名为 ipvlan -net 的额外网络：

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

5.1.5. 配置 macvlan 额外网络

以下对象描述了 MACVLAN CNI 插件的配置参数：

表 5.3. MACVLAN CNI 插件 JSON 配置对象

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

注意

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

5.1.5.1. MACVLAN CNI 插件配置示例

以下示例配置了名为 macvlan-net 的额外网络：

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

5.1.6. 其他资源

- 配置和使用多个网络

5.2. 配置和使用多个网络

安装 MicroShift Multus Container Network Interface (CNI)后，您可以使用配置使用其他网络插件。

5.2.1. IP 地址管理类型和其他网络

通过您配置的 IP 地址管理(IPAM) CNI 插件为额外网络置备 IP 地址。MicroShift 中支持的 IP 地址置备类型是 host-local、static 和 dhcp。

5.2.1.1. 特定于网桥接口的

当使用 网桥 类型接口和 dhcp IPAM 时，需要一个侦听桥接网络的 DHCP 服务器。如果您使用防火墙，请通过运行 firewall-cmd --remove-service=dhcp 命令来配置 firewalld 服务，以便同时允许网络区上的 DHCP 流量。

5.2.1.2. 特定于 macvlan 接口

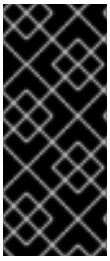
macvlan 类型接口访问主机所连接的网络。这意味着，如果使用 dhcp IPAM 插件，接口可以从主机网络上的 DHCP 服务器接收 IP 地址。

5.2.1.3. IPVLAN 接口相关

ipvlan 接口还直接访问主机网络，但与主机接口共享一个 MAC 地址。由于共享 MAC 地址，ipvlan 类型接口不能与 dhcp 插件一起使用。IPAM 插件不支持带有 ClientID 的 DHCP 协议。

5.2.2. 为额外网络创建 NetworkAttachmentDefinition

使用以下步骤为额外网络创建 NetworkAttachmentDefinition 配置文件。在本例中，使用了 bridge-type 接口。您还可以使用此处的示例工作流，它使用 host-local IP 地址管理(IPAM)来配置其他支持的额外网络类型。



重要

如果您使用 网桥和 dhcp IPAM，则需要侦听桥接网络的 DHCP 服务器。如果您也使用防火墙，请将 firewalld 服务配置为允许网络区中的 DHCP 流量。在这种情况下，您可以运行 firewall-cmd --remove-service=dhcp 命令。

先决条件

- MicroShift Multus CNI 已安装。
- 已安装 OpenShift CLI (oc)。
- 集群正在运行。

流程

1. 可选：运行以下命令来验证 MicroShift 集群是否使用 Multus CNI 运行：

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

输出示例

```
NAME              READY   STATUS    RESTARTS   AGE
dhcp-daemon-dfbzw   1/1     Running   0          5h
multus-rz8xc        1/1     Running   0          5h
```

2. 运行以下命令来创建 NetworkAttachmentDefinition 配置文件，并使用以下示例文件作为参考：

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

NetworkAttachmentDefinition 文件示例

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-conf
spec:
  config: '{
     "cniVersion": "0.4.0",
     "type": "bridge", (1)
     "bridge": "br-test", (2)
     "mode": "bridge",
     "ipam": {
       "type": "host-local", (3)
       "ranges": [
         [
           {
             "subnet": "10.10.0.0/24",
             "rangeStart": "10.10.0.20",
             "rangeEnd": "10.10.0.50",
             "gateway": "10.10.0.254"
           }
         ],
         [
           {
             "subnet": "fd00:IJKL:MNOP:10::0/64", (4)
             "rangeStart": "fd00:IJKL:MNOP:10::1",
             "rangeEnd": "fd00:IJKL:MNOP:10::9"
       "dataDir": "/var/lib/cni/br-test"
     }
   }'
```

(1) type 值指定 CNI 插件的名称。本例使用 bridge 类型。

(2) bridge 值是使用的 MicroShift 主机上的网桥名称。pod 的额外接口连接到该网桥。如果主机上不存在接口，则 Bridge CNI 会创建它。如果接口已存在，它将被重复使用。在本例中，接口名称为 br-test。

(3) IPAM 类型。

(4) IPv6 地址可以添加到二级接口。

注意

使用网桥的名称特定于插件 的网桥 类型。其他插件在其 NetworkAttachmentDefinition 中使用不同的字段。例如，macvlan 和 ipvlan 配置使用 master 指定要附加的主机接口。

5.2.3. 将 pod 添加到额外网络

您可以将 pod 添加到额外网络。在创建 pod 时，会附加额外网络。pod 继续通过默认网络发送与集群相关的普通网络流量。

如果要将额外网络附加到已在运行的 pod，您必须重启 pod。

先决条件

- 已安装 OpenShift CLI (oc)。
- 集群正在运行。
- 您要将 pod 附加到的 NetworkAttachmentDefinition 对象定义的网络。

流程

1. 为 Pod YAML 文件添加注解。只能使用以下注解格式之一：

   a. 要在没有自定义的情况下附加额外网络，请使用以下格式添加注解。将 <network> 替换为要与 pod 关联的额外网络的名称：

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

1. 将 & lt;network > 替换为要与 pod 关联的额外网络的名称。要指定多个额外网络，请使用逗号分隔各个网络。逗号之间不可包括空格。如果您多次指定同一额外网

络，则该 pod 会将多个网络接口附加到该网络。

网桥类型额外网络的注解示例

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
   k8s.v1.cni.cncf.io/networks: bridge-conf
# ...
```

b. 要通过自定义来附加额外网络，请添加具有以下格式的注解：

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
   k8s.v1.cni.cncf.io/networks: |-
    [
     {
      "name": "<network>", 1
      "namespace": "<namespace>", 2
      "default-route": ["<default-route>"] 3
     }
    ]
# ...
```

1 指定 NetworkAttachmentDefinition 对象定义的额外网络的名称。

2 指定定义 NetworkAttachmentDefinition 对象的命名空间。

3 可选：为默认路由指定覆盖，如 192.168.17.1。

2. 要创建 Pod YAML 文件并为额外网络添加 NetworkAttachmentDefinition 注解，请运行以

下命令并使用示例 YAML：

```
$ oc apply -f ./<test-bridge>.yaml 1
```

1

将 *<test-bridge>* 替换为您要使用的 pod 名称。

输出示例

```
pod/test-bridge created
```

test-bridge pod YAML 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: test-bridge
  annotations:
    k8s.v1.cni.cncf.io/networks: bridge-conf
  labels:
    app: test-bridge
spec:
  terminationGracePeriodSeconds: 0
  containers:
  - name: hello-microshift
    image: quay.io/microshift/busybox:1.36
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo -ne \"HTTP/1.0 200 OK\r\nContent-Length:
16\r\n\r\nHello MicroShift\" | nc -l -p 8080 ; done"]
    ports:
    - containerPort: 8080
      protocol: TCP
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
      runAsNonRoot: true
      runAsUser: 1001
      runAsGroup: 1001
      seccompProfile:
        type: RuntimeDefault
```

-

3. 确保 NetworkAttachmentDefinition 注解正确：

NetworkAttachmentDefinition 注解示例

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: bridge-conf
# ...
```

4. 可选： 要确认 Pod YAML 中是否存在 NetworkAttachmentDefinition 注解，请运行以下命令，将 < name& gt; 替换为 pod 的名称。

```
$ oc get pod <name> -o yaml 1
```

1. 将 *<name >* 替换为您要使用的 pod 名称。在以下示例中使用了 test-bridge。

在以下示例中，test-bridge 附加到 net1 额外网络：

```
$ oc get pod test-bridge -o yaml
```

输出示例

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: bridge-conf
    k8s.v1.cni.cncf.io/network-status: |- 1
      [{
```

```
        "name": "ovn-kubernetes",
        "interface": "eth0",
        "ips": [
            "10.42.0.18"
        ],
        "default": true,
        "dns": {}
    },{
        "name": "bridge-conf",
        "interface": "net1",
        "ips": [
            "20.2.2.100"
        ],
        "mac": "22:2f:60:a5:f8:00",
        "dns": {}
    }]
  name: pod
  namespace: default
spec:
# ...
status:
# ...
```

1

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

5. 运行以下命令验证 pod 是否正在运行：

```
$ oc get pod
```

输出示例

```
NAME          READY   STATUS    RESTARTS   AGE
test-bridge   1/1     Running   0          81s
```

5.2.4. 配置额外网络

创建 NetworkAttachmentDefinition 对象并应用它后，使用以下示例步骤来配置额外网络。在本例中，使用了 网桥 类型额外网络。您还可以将此工作流用于其他额外网络类型。

前提条件

1. 您创建并应用 NetworkAttachmentDefinition 对象配置。

流程

1. 运行以下命令验证主机上是否已创建网桥：

```
$ ip a show br-test
```

输出示例

```
22: br-test: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    link/ether 96:bf:ca:be:1d:15 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::34e2:bbff:fed2:31f2/64 scope link
       valid_lft forever preferred_lft forever
```

2. 运行以下命令，为网桥配置 IP 地址：

```
$ sudo ip addr add 10.10.0.10/24 dev br-test
```

3. 运行以下命令，验证 IP 地址配置是否已添加到桥接中：

```
$ ip a show br-test
```

输出示例

```
22: br-test: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    link/ether 96:bf:ca:be:1d:15 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.10/24 scope global br-test 1
```

```
      valid_lft forever preferred_lft forever
    inet6 fe80::34e2:bbff:fed2:31f2/64 scope link
      valid_lft forever preferred_lft forever
```

1

IP 地址按预期配置。

4. 运行以下命令，验证 pod 的 IP 地址：

```
$ oc get pod test-bridge --
output=jsonpath='{.metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status}'
```

输出示例

```
[{
    "name": "ovn-kubernetes",
    "interface": "eth0",
    "ips": [
        "10.42.0.17"
    ],
    "mac": "0a:58:0a:2a:00:11",
    "default": true,
    "dns": {}
},{
    "name": "default/bridge-conf", 1
    "interface": "net1",
    "ips": [
        "10.10.0.20"
    ],
    "mac": "82:01:98:e5:0c:b7",
    "dns": {}
```

1

网桥额外网络会如预期附加。

-

5. 可选： 您可以使用 oc exec 访问 pod 并使用 ip 命令确认其接口：

```
$ oc exec -ti test-bridge -- ip a
```

输出示例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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
2: eth0@if21: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
    link/ether 0a:58:0a:2a:00:11 brd ff:ff:ff:ff:ff:ff
    inet 10.42.0.17/24 brd 10.42.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::858:aff:fe2a:11/64 scope link
       valid_lft forever preferred_lft forever
3: net1@if23: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
    link/ether 82:01:98:e5:0c:b7 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.20/24 brd 10.10.0.255 scope global net1 (1)
       valid_lft forever preferred_lft forever
    inet6 fe80::8001:98ff:fee5:cb7/64 scope link
       valid_lft forever preferred_lft forever
```

(1) Pod 按预期附加到 net 1 接口上的 10.10.0.20 IP 地址。

6. 通过从 MicroShift 主机访问 pod 中的 HTTP 服务器来确认连接按预期工作。使用以下命令：

```
$ curl 10.10.0.20:8080
```

输出示例

```
Hello MicroShift
```

5.2.5. 从额外网络中删除 pod

您只能通过删除 pod 来从额外网络中删除 pod。

先决条件

- 一个额外网络被附加到 pod。
- 安装 OpenShift CLI（oc）。
- 登录到集群。

流程

- 要删除 pod，输入以下命令：

```
$ oc delete pod <name> -n <namespace>
```

  - <name> 是 pod 的名称。
  - <namespace> 是包含 pod 的命名空间。

5.2.6. Multus 网络故障排除

如果没有正确配置多个网络的设置，pod 可能无法启动。以下步骤可帮助您解决几个常见情况。

5.2.6.1. 无法配置 Pod 网络

如果 Multus CNI 插件无法将网络注解应用到 pod，则 pod 不会启动。如果任何额外网络 CNI 失败，Pod 也无法启动。

错误示例

```
Warning  NoNetworkFound  0s  multus  cannot find a network-attachment-definitio
(asdasd) in namespace (default): network-attachment-definitions.k8s.cni.cncf.io "bad-ref-
doesnt-exist" not found
```

在这种情况下，您可以执行以下步骤解决 CNI 失败：

- 验证 NetworkAttachmentDefinition 和注解 中的值。
- 删除注解，以验证 pod 是否使用默认网络成功创建。如果没有，这可能表示 Multus 配置以外的网络问题。
- 如果您是设备管理员，您可以检查 crio.service 或 microshift.service 日志，请特别注意 kubelet 生成的内容。

  例如，kubelet 的以下错误显示主 CNI 未运行。这种情况可能是由 pod 启动或因为 CRI-O 错误配置导致，如不正确的 cni_default_network 设置。

  kubelet 生成的错误示例

```
Feb 06 13:47:31 dev microshift[1494]: kubelet E0206 13:47:31.163290  1494
pod_workers.go:1298] "Error syncing pod, skipping" err="network is not ready:
container runtime network not ready: NetworkReady=false
reason:NetworkPluginNotReady message:Network plugin returns error: No CNI
configuration file in /etc/cni/net.d/. Has your network provider started?"
pod="default/samplepod" podUID="fe0f7f7a-8c47-4488-952b-8abc0d8e2602"
```

5.2.6.2. 缺少的配置文件

有时无法创建 pod，因为注解引用不存在的 NetworkAttachmentDefinition 配置 YAML。在这种情况下，通常会生成类似如下的错误：

日志示例

```
cannot find a network-attachment-definition (bad-conf) in namespace (default): network-
attachment-definitions.k8s.cni.cncf.io "bad-conf" not found" pod="default/samplepod"`
```

错误输出示例

```
"CreatePodSandbox for pod failed" err="rpc error: code = Unknown desc = failed to create
pod network sandbox k8s_samplepod_default_5fa13105-1bfb-4c6b-aee7-
3437cfb50e25_0(7517818bd8e85f07b551f749c7529be88b4e7daef0dd572d049aa636950c76c6):
error adding pod default_samplepod to CNI network \"multus-cni-network\": plugin
type=\"multus\" name=\"multus-cni-network\" failed (add): Multus:
[default/samplepod/5fa13105-1bfb-4c6b-aee7-3437cfb50e25]: error loading k8s delegates k8s
args: TryLoadPodDelegates: error in getting k8s network for pod: GetNetworkDelegates:
failed getting the delegate: getKubernetesDelegate: cannot find a network-attachment-
definition (bad-conf) in namespace (default): network-attachment-definitions.k8s.cni.cncf.io
\"bad-conf\" not found" pod="default/samplepod"
```

要修复此错误，请创建并应用 NetworkAttachmentDefinition YAML。

5.2.7. 其他资源

- 关于使用多个网络
- 为额外网络配置 IP 地址分配

第 6 章 配置路由

您可以为集群配置 MicroShift 的路由。

6.1. 创建基于 HTTP 的路由

路由允许您在公共 URL 托管应用程序。根据应用程序的网络安全配置，它可以安全或不受保护。基于 HTTP 的路由是一个不受保护的路由，它使用基本的 HTTP 路由协议，并在未安全的应用程序端口上公开服务。

以下流程描述了如何使用 hello-openshift 应用程序创建基于 HTTP 的简单路由，作为示例。

先决条件

- 已安装 OpenShift CLI（oc）。
- 以管理员身份登录。
- 您有一个 web 应用，用于公开端口和侦听端口上流量的 TCP 端点。

流程

1. 运行以下命令，创建一个名为 hello-openshift 的项目：

```
$ oc new-project hello-openshift
```

2. 运行以下命令，在项目中创建 pod：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-
openshift/hello-pod.json
```

3. 运行以下命令，创建名为 hello-openshift 的服务：

```
$ oc expose pod/hello-openshift
```

4. 运行以下命令，创建一个没有安全安全的路由到 hello-openshift 应用程序：

```
$ oc expose svc hello-openshift
```

验证

- 要验证您创建的路由资源，请运行以下命令：

```
$ oc get routes -o yaml <name of resource> 1
```

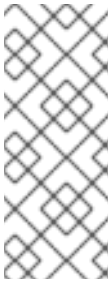
1 在本例中，路由名为 hello-openshift。

创建的未安全路由的 YAML 定义示例：

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain> 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

1 <Ingress_Domain> 是默认的入口域名。ingresses.config/cluster 对象是在安装过程中创建的，且无法更改。如果要指定不同的域，您可以使用 appsDomain 选项指定备选集群域。

2 targetPort 是由此路由指向的服务选择的 pod 上的目标端口。



注意

要显示您的默认入口域，请运行以下命令：

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

6.1.1. HTTP 严格传输安全性

HTTP 严格传输安全性 (HSTS) 策略是一种安全增强，向浏览器客户端发送信号，表示路由主机上仅允许 HTTPS 流量。HSTS 也通过信号 HTTPS 传输来优化 Web 流量，无需使用 HTTP 重定向。HSTS 对于加快与网站的交互非常有用。

强制 HSTS 策略时，HSTS 会向站点的 HTTP 和 HTTPS 响应添加 Strict Transport Security 标头。您可以在路由中使用 insecureEdgeTerminationPolicy 值，以将 HTTP 重定向到 HTTPS。强制 HSTS 时，客户端会在发送请求前将所有请求从 HTTP URL 更改为 HTTPS，无需重定向。

集群管理员可将 HSTS 配置为执行以下操作：

- 根据每个路由启用 HSTS
- 根据每个路由禁用 HSTS
- 对一组域强制每个域的 HSTS，或者结合使用命名空间标签与域

重要

HSTS 仅适用于安全路由，可以是 edge-terminated 或 re-encrypt。其配置在 HTTP 或传递路由上无效。

6.1.2. 根据每个路由启用 HTTP 严格传输安全性

HTTP 严格传输安全 (HSTS) 实施在 HAProxy 模板中，并应用到具有 haproxy.router.openshift.io/hsts_header 注解的边缘和重新加密路由。

先决条件

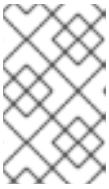
- 有对集群的 root 访问权限。
- 已安装 OpenShift CLI（oc）。

流程

- 要在路由上启用 HSTS，请将 haproxy.router.openshift.io/hsts_header 值添加到 edge-terminated 或 re-encrypt 路由中。您可以运行以下命令来使用 oc annotate 工具来实现此目的：

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000;\ 1
includeSubDomains;preload"
```

1. 在本例中，最长期限设置为 31536000 ms，大约为 8.5 小时。



注意

在这个示例中，等号 (=) 包括在引号里。这是正确执行注解命令所必需的。

配置了注解的路由示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-
age=31536000;includeSubDomains;preload 1 2 3
...
spec:
  host: def.abc.com
  tls:
    termination: "reencrypt"
    ...
  wildcardPolicy: "Subdomain"
```

1

必需。Max-age 测量 HSTS 策略生效的时间长度，以秒为单位。如果设置为 0，它将对策略进行求反。

2

可选。包含时，includeSubDomains 告知客户端主机的所有子域都必须与主机具有相同的 HSTS 策略。

3

可选。当 max-age 大于 0 时，您可以在 haproxy.router.openshift.io/hsts_header 中添加 preload，以允许外部服务将这个站点包括在 HSTS 预加载列表中。例如，Google 等站点可以构造设有 preload 的站点的列表。浏览器可以使用这些列表来确定哪些站点可以通过 HTTPS 通信，即使它们与站点交互之前也是如此。如果没有设置 preload，浏览器必须已经通过 HTTPS 与站点交互（至少一次）才能获取标头。

6.1.3. 根据每个路由禁用 HTTP 严格传输安全性

要禁用 HTTP 严格传输安全性 (HSTS)，您可以将路由注解中的 max-age 值设置为 0。

先决条件

- 有对集群的 root 访问权限。
- 已安装 OpenShift CLI（oc）。

流程

- 要禁用 HSTS，请输入以下命令将路由注解中的 max-age 值设置为 0 ：

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

提示

您还可以应用以下 YAML 来创建配置映射：

根据每个路由禁用 HSTS 的示例

```
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- 要为命名空间中的所有路由禁用 HSTS，请输入以下命令：

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

验证

1. 要查询所有路由的注解，请输入以下命令：

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if
.metadata.annotations}}{{$a := index .metadata.annotations
"haproxy.router.openshift.io/hsts_header"}}{{$n := .metadata.name}}{{with $a}}Name:
{{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
```

输出示例

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

6.1.4. 强制每个域 HTTP 严格传输安全性

您可以使用兼容 HSTS 策略注解配置路由。要使用不合规的 HSTS 路由处理升级的集群，您可以在源更新清单并应用更新。

您不能使用 oc expose route 或 oc create route 命令在强制 HSTS 的域中添加路由，因为这些命令的 API 不接受注解。

重要

HSTS 无法应用到不安全或非 TLS 路由。

先决条件

- 有对集群的 root 访问权限。
- 已安装 OpenShift CLI（oc）。

流程

- 运行以下 oc annotate command 将 HSTS 应用到集群中的所有路由：

```
$ oc annotate route --all --all-namespaces --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-
age=31536000;preload;includeSubDomains"
```

- 通过运行以下 oc annotate 命令，将 HSTS 应用到特定命名空间中的所有路由：

```
$ oc annotate route --all -n <my_namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-
age=31536000;preload;includeSubDomains" 1
```

1

将 _<my_namespace> 替换为您要使用的命名空间。

验证

- 运行以下命令，查看所有路由上的 HSTS 注解：

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if
.metadata.annotations}}{{$a := index .metadata.annotations
"haproxy.router.openshift.io/hsts_header"}}{{$n := .metadata.name}}{{with $a}}Name:
```

```
{{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
```

输出示例

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

6.2. 吞吐量问题的故障排除方法

有时，使用红帽 MicroShift 部署的应用程序可能会导致网络吞吐量问题，如特定服务间的延迟异常高。

如果 pod 日志没有显示造成问题的原因，请使用以下方法之一分析性能问题：

- 使用 ping 或 tcpdump 等数据包分析器，分析 pod 与其节点之间的流量。

  例如，在每个 pod 上运行 tcpdump 工具，同时重现导致问题的行为。检查两端的捕获信息，以便比较发送和接收时间戳来分析与 pod 往来的流量的延迟。如果节点接口被其他 pod、存储设备或数据平面的流量过载，则红帽构建的 MicroShift 中可能会出现延迟。

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> 1
```

1

  podip 是 pod 的 IP 地址。运行 oc get pod <pod_name> -o wide 命令来获取 pod 的 IP 地址。

  tcpdump 命令会在 /tmp/dump.pcap 中生成一个包含这两个 pod 间所有流量的文件。您可以在运行分析器后立即重现问题，并在问题重现完成后马上停止分析器，从而尽量减小文件的大小。您还可以通过以下命令，在节点之间运行数据包分析器（从考量范围中剔除 SDN）：

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- 使用 iperf 等带宽测量工具来测量流吞吐量和 UDP 吞吐量。首先从 pod 运行该工具，然后从节点运行它，从而找到瓶颈。

6.3. 使用 COOKIE 来保持路由有状态性

MicroShift 的红帽构建提供粘性会话，通过确保所有流量都到达同一端点来实现有状态应用程序流量。但是，如果端点 pod 以重启、扩展或更改配置的方式被终止，这种有状态性可能会消失。

红帽构建的 MicroShift 可使用 Cookie 来配置会话持久性。ingress 控制器选择一个端点来处理任何用户请求，并为会话创建一个 Cookie。Cookie 在响应请求时返回，用户则通过会话中的下一请求发回 Cookie。Cookie 告知入口控制器处理会话，确保客户端请求使用这个 Cookie 使请求路由到同一个 pod。

注意

无法在 passthrough 路由上设置 Cookie，因为无法看到 HTTP 流量。相反，根据源 IP 地址计算数字，该地址决定了后端。

如果后端更改，可以将流量定向到错误的服务器，使其更不计。如果您使用负载均衡器来隐藏源 IP，则会为所有连接和流量都发送到同一 pod 设置相同的数字。

6.3.1. 使用 Cookie 标注路由

您可以设置 Cookie 名称来覆盖为路由自动生成的默认名称。这样，接收路由流量的应用程序就能知道 Cookie 名称。删除 Cookie 可强制下一请求重新选择端点。结果是，如果服务器过载，该服务器会尝试从客户端中删除请求并重新分发它们。

流程

1. 使用指定的 Cookie 名称标注路由：

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

其中：

<route_name>

指定路由的名称。

<cookie_name>

指定 Cookie 的名称。

例如，使用 cookie 名称 my_cookie 标注路由 my_route：

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. 在变量中捕获路由主机名：

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

其中：

<route_name>

指定路由的名称。

3. 保存 cookie，然后访问路由：

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

使用上一个命令在连接到路由时保存的 cookie：

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

6.4. 基于路径的路由

基于路径的路由指定了一个路径组件，可以与 URL 进行比较，该 URL 需要基于 HTTP 的路由流量。因此，可以使用同一主机名提供多个路由，每个主机名都有不同的路径。路由器应该匹配基于最具体路径的路由。

下表显示了路由及其可访问性示例：

表 6.1. 路由可用性

Route（路由）	当比较到	可访问
www.example.com/test	*www.example.com/test*	是
	www.example.com	否

Route（路由）	当比较到	可访问
www.example.com/test 和 *www.example.com*	*www.example.com/test*	是
	www.example.com	是
www.example.com	*www.example.com/text*	yes（由主机匹配，而不是路由）
	www.example.com	是

带有路径的未安全路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
  to:
    kind: Service
    name: service-name
```

1

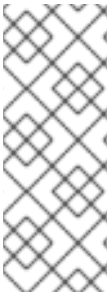
该路径是基于路径的路由的唯一添加属性。

注意

使用 passthrough TLS 时，基于路径的路由不可用，因为路由器不会在这种情况下终止 TLS，且无法读取请求的内容。

6.5. HTTP 标头配置

在设置或删除标头时，您可以使用单个路由来修改请求和响应标头。您还可以使用路由注解设置某些标头。配置标头的各种方法在协同工作时可能会带来挑战。



注意

您只能在 Route CR 中设置或删除标头。您无法附加标头。如果使用值设置 HTTP 标头，则该值必须已完成，且在以后不需要附加。在附加标头（如 X-Forwarded-For 标头）时，请使用 spec.httpHeaders.forwardedHeaderPolicy 字段，而不是 spec.httpHeaders.actions。

Route 规格示例

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: SAMEORIGIN
```

使用路由注解设置的路由覆盖值中定义的任何操作。

6.5.1. 特殊情况标头

以下标头可能会阻止完全被设置或删除，或者在特定情况下允许：

标头名称	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
proxy	否	**proxy** HTTP 请求标头可以通过将标头值注入 **HTTP_PROXY** 环境变量来利用这个安全漏洞的 CGI 应用程序。**proxy** HTTP 请求标头也是非标准的，在配置期间容易出错。	否

标头名称	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
主机	是	当使用 **IngressController** CR 设置 **host** HTTP 请求标头时，HAProxy 在查找正确的路由时可能会失败。	否
strict-transport-security	否	**strict-transport-security** HTTP 响应标头已使用路由注解处理，不需要单独的实现。	是：**haproxy.router.openshift.io/hsts_header** 路由注解
cookie 和 **set-cookie**	否	HAProxy 集的 Cookie 用于会话跟踪，用于将客户端连接映射到特定的后端服务器。允许设置这些标头可能会影响 HAProxy 的会话关联，并限制 HAProxy 的 Cookie 的所有权。	是： * **haproxy.router.openshift.io/disable_cookie** 路由注解 * **haproxy.router.openshift.io/cookie_name** 路由注解

6.6. 在路由中设置或删除 HTTP 请求和响应标头

出于合规的原因，您可以设置或删除某些 HTTP 请求和响应标头。您可以为 Ingress Controller 提供的所有路由或特定路由设置或删除这些标头。

例如，如果内容使用多种语言编写，您可能希望让 Web 应用程序在备用位置提供内容，即使 Ingress Controller 为路由指定的默认全局位置。

以下流程会创建一个设置 Content-Location HTTP 请求标头的路由，以便与应用程序关联的 URL https://app.example.com 定向到位置 https://app.example.com/lang/en-us。将应用程序流量定向到此位置意味着使用该特定路由的任何人都可以访问以美国英语编写的 Web 内容。

先决条件

- 已安装 OpenShift CLI（oc）。

- 以项目管理员身份登录到 MicroShift 集群的红帽构建。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。

流程

1. 创建一个路由定义，并将它保存到名为 app-example-route.yaml 的文件中：

使用 HTTP 标头指令创建路由的 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
    name: app-example
  httpHeaders:
    actions: 1
      response: 2
      - name: Content-Location 3
        action:
          type: Set 4
          set:
            value: /lang/en-us 5
```

1. 要在 HTTP 标头上执行的操作列表。
2. 您要更改的标头类型。在本例中，响应标头。
3.

您要更改的标头的名称。有关您可以设置或删除的可用标头列表，请参阅 *HTTP 标头配置*。

4

在标头中执行的操作类型。此字段可以具有 Set 或 Delete 的值。

5

在设置 HTTP 标头时，您必须提供一个 value。该值可以是该标头的可用指令列表中的字符串，如 DENY，也可以是使用 HAProxy 的动态值语法来解释的动态值。在这种情况下，该值被设置为内容的相对位置。

2. 使用新创建的路由定义，创建到现有 Web 应用程序的路由：

```
$ oc -n app-example create -f app-example-route.yaml
```

对于 HTTP 请求标头，路由定义中指定的操作会在 Ingress Controller 中对 HTTP 请求标头执行的任何操作后执行。这意味着，路由中这些请求标头设置的任何值都将优先于 Ingress Controller 中设置的值。有关 HTTP 标头处理顺序的更多信息，请参阅 *HTTP 标头配置*。

6.7. 通过 INGRESS 对象创建路由

一些生态系统组件与 Ingress 资源集成，但与路由资源不集成。为了涵盖这种情况，红帽构建的 MicroShift 会在创建 Ingress 对象时自动创建受管路由对象。当相应 Ingress 对象被删除时，这些路由对象会被删除。

流程

1. 在 MicroShift 控制台的 Red Hat build 中或通过 oc create 命令定义 Ingress 对象：

Ingress 的 YAML 定义

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt" 1
```

```
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert (2)
spec:
  rules:
  - host: www.example.com (3)
    http:
      paths:
      - backend:
          service:
            name: frontend
            port:
              number: 443
        path: /
        pathType: Prefix
  tls:
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate
```

(1)

route.openshift.io/termination 注解可用于配置 Route 的 spec.tls.termination 字段，因为 Ingress 没有此字段。可接受的值为 edge、passthrough 和 reencrypt。所有其他值都会被静默忽略。当注解值未设置时，edge 是默认路由。模板文件中必须定义 TLS 证书详细信息，才能实现默认的边缘路由。

(3)

在使用 Ingress 对象时，您必须指定一个显式主机名，这与使用路由时不同。您可以使用 <host_name>.<cluster_ingress_domain> 语法（如 apps.openshiftdemos.com）以利用 *.<cluster_ingress_domain> 通配符 DNS 记录，为集群提供证书。否则，您必须确保有一个用于所选主机名的 DNS 记录。

a.

如果您在 route.openshift.io/termination 注解中指定 passthrough 值，在 spec 中将 path 设置为 ''，将 pathType 设置为 ImplementationSpecific :

```
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: ''
        pathType: ImplementationSpecific
        backend:
          service:
            name: frontend
            port:
              number: 443
```

-

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

2

route.openshift.io/destination-ca-certificate-secret 可用于 Ingress 对象来定义带有自定义目的地证书(CA)的路由。该注解引用一个 kubernetes secret，secret-ca-cert 将插入到生成的路由中。

a. 要从 ingress 对象使用目标 CA 指定路由对象，您必须在 secret 的 data.tls.crt specifier 中创建一个带有 PEM 编码格式的证书的 kubernetes.io/tls 或 Opaque 类型 secret。

2. 列出您的路由：

```
$ oc get routes
```

结果包括一个自动生成的路由，其名称以 frontend- 开头：

```
NAME              HOST/PORT         PATH   SERVICES   PORT   TERMINATION   WILDCARD
frontend-gnztq    www.example.com          frontend   443    reencrypt/Redirect   None
```

如果您检查这个路由，它会类似于：

自动生成的路由的 YAML 定义

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
  ownerReferences:
  - apiVersion: networking.k8s.io/v1
    controller: true
    kind: Ingress
    name: frontend
    uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
```

```
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      [...]
      -----END RSA PRIVATE KEY-----
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
  to:
    kind: Service
    name: frontend
```

6.8. 通过 INGRESS 对象使用默认证书创建路由

如果您在没有指定 TLS 配置的情况下创建 Ingress 对象，红帽构建的 MicroShift 会生成不安全的路由。要创建使用默认入口证书生成安全边缘终止路由的 Ingress 对象，您可以指定一个空的 TLS 配置，如下所示：

先决条件

- 您有一个要公开的服务。
- 您可以访问 OpenShift CLI(oc)。

流程

1. 为 Ingress 对象创建 YAML 文件。在本例中，该文件名为 example-ingress.yaml ：

Ingress 对象的 YAML 定义

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
```

```
  ...
spec:
  rules:
    ...
  tls:
  - {} 1
```

1

使用此精确的语法指定 TLS，而不指定自定义证书。

2. 运行以下命令来创建 Ingress 对象：

```
$ oc create -f example-ingress.yaml
```

验证

- 运行以下命令，验证红帽构建的 MicroShift 是否为 Ingress 对象创建了预期的路由：

```
$ oc get routes -o yaml
```

输出示例

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd 1
    ...
  spec:
  ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
  ...
```

1. 路由的名称包括 Ingress 对象的名称，后跟一个随机的后缀。

2. 要使用默认证书，路由不应指定 spec.certificate。

3. 路由应指定 edge 终止策略。

6.9. 在 INGRESS 注解中使用目标 CA 证书创建路由

在 Ingress 对象上可以使用 route.openshift.io/destination-ca-certificate-secret 注解来定义带有自定义目标 CA 证书的路由。

先决条件

- 您可以在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须在 PEM 编码文件中有单独的目标 CA 证书。
- 您必须具有要公开的服务。

流程

1. 将 route.openshift.io/destination-ca-certificate-secret 添加到 Ingress 注解中：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert (1)
...
```

1 该注解引用 kubernetes secret。

2. 此注解中引用的机密将插入到生成的路由中。

输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

6.10. 安全路由

安全路由提供以下几种 TLS 终止功能来为客户端提供证书。以下到 OpenShift Container Platform 文档的链接描述了如何使用自定义证书创建重新加密、边缘和透传路由。

- 使用自定义证书创建重新加密路由
- 使用自定义证书创建边缘路由
- 创建 passthrough 路由

第 7 章 使用防火墙

MicroShift 中不需要防火墙，但使用防火墙可以防止对 MicroShift API 的不必要的访问权限。

7.1. 关于通过防火墙的网络流量

firewalld 是一个在后台运行并响应连接请求的网络服务，创建基于主机的动态自定义防火墙。如果您使用带有 MicroShift 的 Red Hat Enterprise Linux for Edge (RHEL for Edge)，则应该已安装 firewalld，您只需要配置它。遵循的步骤中提供了详细信息。总体而言，当 firewalld 服务运行时，您必须明确允许以下 OVN-Kubernetes 流量：

CNI pod 到 CNI pod

CNI pod 到 Host-Network pod Host-Network pod 到 Host-Network pod

CNI pod

使用 CNI 网络的 Kubernetes pod

Host-Network pod

使用主机网络的 Kubernetes pod，您可以按照以下步骤配置 firewalld 服务。在大多数情况下，firewalld 是 RHEL for Edge 安装的一部分。如果没有 firewalld，您可以参照本节中的简单流程安装它。

重要

MicroShift pod 必须有权访问内部 CoreDNS 组件和 API 服务器。

其他资源

- 所需的防火墙设置
- 允许通过防火墙的网络流量

7.2. 安装 FIREWALLD 服务

如果您使用 RHEL for Edge，则应该安装 firewalld。要使用该服务，您可以简单地进行配置。如果您没有 firewalld，但希望使用它，则可以使用以下步骤。

使用以下步骤为 MicroShift 安装并运行 firewalld 服务。

流程

1. 可选：运行以下命令来检查系统中的 firewalld ：

```
$ rpm -q firewalld
```

2. 如果没有安装 firewalld 服务，请运行以下命令：

```
$ sudo dnf install -y firewalld
```

3. 要启动防火墙，请运行以下命令：

```
$ sudo systemctl enable firewalld --now
```

7.3. 所需的防火墙设置

在防火墙配置过程中，必须启用集群网络的 IP 地址范围。您可以使用默认值或自定义 IP 地址范围。如果您选择从默认的 10.42.0.0/16 设置自定义集群网络 IP 地址范围，还必须在防火墙配置中使用相同的自定义范围。

表 7.1. 防火墙 IP 地址设置

IP 范围	防火墙规则所需的	描述
10.42.0.0/16	否	主机网络 pod 访问其他 pod
169.254.169.1	是	主机网络 pod 访问红帽构建的 MicroShift API 服务器

以下是防火墙配置强制设置的命令示例：

示例命令

- 配置主机网络 pod 对其他 pod 的访问：

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=10.42.0.0/16
```

-

- 配置主机网络 pod 访问由主机端点支持的服务，如红帽 MicroShift API 的构建：

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=169.254.169.1
```

7.4. 使用可选端口设置

MicroShift 防火墙服务允许可选端口设置。

流程

- 要在防火墙配置中添加自定义端口，请使用以下命令语法：

```
$ sudo firewall-cmd --permanent --zone=public --add-port=<port number>/<port protocol>
```

表 7.2. 可选端口

端口	协议	描述
80	TCP	用于通过 OpenShift Container Platform 路由器提供应用程序的 HTTP 端口。
443	TCP	用于通过 OpenShift Container Platform 路由器提供应用程序的 HTTPS 端口。
5353	UDP	mDNS 服务以响应 OpenShift Container Platform 路由 mDNS 主机。
30000-32767	TCP	为 NodePort 服务保留的端口范围；可用于公开 LAN 上的应用程序。
30000-32767	UDP	为 NodePort 服务保留的端口范围；可用于公开 LAN 上的应用程序。
6443	TCP	用于红帽构建的 MicroShift API 的 HTTPS API 端口。

以下是当要求从外部访问在 MicroShift 上运行的服务时使用的命令示例，如 API 服务器的端口 6443，

例如通过路由器公开的应用程序的端口 80 和 443。

示例命令

- 为 MicroShift API 服务器配置端口：

```
$ sudo firewall-cmd --permanent --zone=public --add-port=6443/tcp
```

要关闭 MicroShift 实例中不必要的端口，请按照 "Closing unused 或 unnecessary port to enhance network security" 中的步骤操作。

其他资源

- 关闭未使用的或不必要的端口，来增强网络安全性

7.5. 添加服务来打开端口

在 MicroShift 实例中，您可以使用 firewall-cmd 命令在端口上打开服务。

流程

1. 可选：您可以通过运行以下命令来查看 firewalld 中的所有预定义服务

```
$ sudo firewall-cmd --get-services
```

2. 要在默认端口中打开您想要的服务，请运行以下命令：

```
$ sudo firewall-cmd --add-service=mdns
```

7.6. 允许通过防火墙的网络流量

您可以通过配置 IP 地址范围并插入 DNS 服务器来允许网络流量通过防火墙，以允许通过网络网关从 pod 的内部流量。

流程

1. 使用以下命令之一设置 IP 地址范围：

   a. 运行以下命令，使用默认值配置 IP 地址范围：

   ```
   $ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=10.42.0.0/16
   ```

   b. 运行以下命令，使用自定义值配置 IP 地址范围：

   ```
   $ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=<custom IP range>
   ```

2. 要允许 pod 通过网络网关的内部流量，请运行以下命令：

   ```
   $ sudo firewall-offline-cmd --permanent --zone=trusted --add-source=169.254.169.1
   ```

7.6.1. 应用防火墙设置

要应用防火墙设置，请使用以下步骤步骤：

流程

- 通过防火墙配置网络访问后，运行以下命令重启防火墙并应用设置：

```
$ sudo firewall-cmd --reload
```

7.7. 验证防火墙设置

重启防火墙后，您可以通过列出设置来验证设置。

流程

- 要验证在默认公共区（如端口相关的规则）中添加的规则，请运行以下命令：

  ```
  $ sudo firewall-cmd --list-all
  ```

- 要验证在可信区（如 IP-range 相关规则）中添加的规则，请运行以下命令：

```
$ sudo firewall-cmd --zone=trusted --list-all
```

7.8. 公开服务时防火墙端口概述

当您在 MicroShift 上运行服务时，firewalld 通常处于活跃状态。这可能会中断 MicroShift 上的某些服务，因为防火墙可能会阻止到端口的流量。如果您希望从主机外部访问某些服务，则必须确保打开所需的防火墙端口。打开端口有几个选项：

- NodePort 和 LoadBalancer 类型的服务会自动用于 OVN-Kubernetes。

  在这些情况下，OVN-Kubernetes 添加 iptables 规则，以便到节点 IP 地址的流量传送到相关端口。这使用 PREROUTING 规则链完成，然后转发到 OVN-K 以绕过本地主机端口和服务的 firewalld 规则。iptables 和 firewalld 由 RHEL 9 中的 nftables 支持。iptables 生成的 nftables 规则始终优先于 firewalld 生成的规则。

- 具有 HostPort 参数设置的 Pod 会自动可用。这还包括 router-default pod，它使用端口 80 和 443。

  对于 HostPort pod，CRI-O 配置将 iptables DNAT (Destination Network Address Translation)设置为 pod 的 IP 地址和端口。

这些方法对于客户端是位于同一主机还是远程主机上的功能。OVN-Kubernetes 和 CRI-O 添加的 iptables 规则附加到 PREROUTING 和 OUTPUT 链。本地流量通过 OUTPUT 链，接口设置为 lo 类型。DNAT 在它到达 INPUT 链中的填充规则之前运行。

由于 MicroShift API 服务器没有在 CRI-O 中运行，所以受防火墙配置约束。您可以在防火墙中打开端口 6443，以访问 MicroShift 集群中的 API 服务器。

7.9. 其他资源

- RHEL：使用和配置 firewalld
- RHEL：查看 firewalld 的当前状态

7.10. 已知的防火墙问题

- 为了避免在重新加载或重启的情况下破坏流量流，请在启动 RHEL 前执行防火墙命令。MicroShift 中的 CNI 驱动程序将 iptable 规则用于某些流量流，如使用 NodePort 服务的用户。iptable 规则由 CNI 驱动程序生成和插入，但在防火墙重新加载或重启时会被删除。缺少 iptable 规则会破坏流量流。如果需要在 MicroShift 运行后执行 firewall 命令，请手动重启 openshift-ovn-kubernetes 命名空间中的 ovnkube-master pod 来重置由 CNI 驱动程序控制的规则。

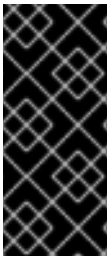
第 8 章 为完全断开连接的主机配置网络设置

了解如何应用网络自定义和设置，以便在完全断开连接的主机上运行 MicroShift。断开连接的主机应该是 Red Hat Enterprise Linux (RHEL)操作系统，版本 9.0+，无论是实际还是虚拟，在没有网络连接的情况下运行。

8.1. 为完全断开连接的主机准备网络

使用以下步骤在运行完全断开连接的操作系统的设备上启动并运行 MicroShift 集群。如果没有外部网络连接，MicroShift 主机被视为完全断开连接的。

通常，这意味着设备没有附加的网络接口控制器(NIC)来提供子网。这些步骤也可以在设置后删除的 NIC 的主机上完成。您还可以使用 Kickstart 文件的 %post 阶段在没有 NIC 的主机上自动化这些步骤。



重要

需要为断开连接的环境配置网络设置，因为 MicroShift 需要网络设备来支持集群通信。要满足此要求，您必须配置 MicroShift 网络设置，以使用您在设置过程中分配给系统回送设备的 "fake" IP 地址。

8.1.1. 步骤概述

要在断开连接的主机上运行 MicroShift，需要以下步骤：

准备主机

- 如果当前正在运行，请停止 MicroShift，并清理该服务已对网络进行了更改。
- 设置持久主机名。
- 在回环接口上添加"fake"IP 地址。
- 将 DNS 配置为使用假的 IP 作为本地名称服务器。
- 向 /etc/hosts 添加主机名条目。

更新 MicroShift 配置

- 将 nodeIP 参数定义为新的回环 IP 地址。
- 将 .node.hostnameOverride 参数设置为持久主机名。

要使更改生效

- 如果附加，禁用默认 NIC。
- 重启主机或设备。

启动后，MicroShift 使用 loopback 设备进行集群内通信运行。

8.2. 将 MICROSHIFT 网络设置恢复到默认值

您可以通过停止 MicroShift 并运行清理脚本来删除网络并将网络返回到默认设置。

先决条件

- RHEL 9 或更新版本。
- MicroShift 4.14 或更新版本。
- 访问主机 CLI。

流程

1. 运行以下命令来停止 MicroShift 服务：

```
$ sudo systemctl stop microshift
```

2. 运行以下命令停止 kubepods.slice systemd 单元：

```
$ sudo systemctl stop kubepods.slice
```

3. MicroShift 会安装帮助程序脚本，以撤销 OVN-K 所做的网络更改。输入以下命令运行清理脚本：

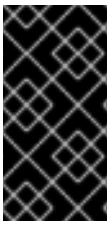
```
$ sudo /usr/bin/microshift-cleanup-data --ovn
```

8.3. 为完全断开连接的主机配置网络设置

要配置网络设置，以便在完全断开连接的主机上运行 MicroShift，您必须准备主机，更新网络配置，然后重启以应用新设置。所有命令都通过主机 CLI 执行。

先决条件

- RHEL 9 或更新版本。
- MicroShift 4.14 或更新版本。
- 访问主机 CLI。
- 选择一个有效的 IP 地址以避免在运行 MicroShift 时出现内部和外部 IP 冲突。
- MicroShift 网络设置设置为默认值。



重要

以下流程适用于在设备部署到字段中后不需要访问 MicroShift 集群的用例。删除网络连接后没有远程集群访问。

流程

1. 运行以下命令，在回环接口中添加假的 IP 地址：

```
$ IP="10.44.0.1" 1
$ sudo nmcli con add type loopback con-name stable-microshift ifname lo ip4 ${IP}/32
```

1 本例中使用的假的 IP 地址是 "10.44.0.1"。

注意

如果避免内部 MicroShift 和潜在的外部 IP 冲突，则任何有效的 IP 都可以正常工作。这可以是不与 MicroShift 节点子网冲突的任何子网，或者可由设备上的其他服务访问。

2. 通过修改设置来忽略自动 DNS 并将其重置为本地名称服务器，将 DNS 接口配置为使用本地名称服务器：

   a. 运行以下命令来绕过自动 DNS：

   ```
   $ sudo nmcli conn modify stable-microshift ipv4.ignore-auto-dns yes
   ```

   b. 将 DNS 接口指向使用本地名称服务器：

   ```
   $ sudo nmcli conn modify stable-microshift ipv4.dns "10.44.1.1"
   ```

3. 运行以下命令来获取该设备的主机名：

   ```
   $ NAME="$(hostnamectl hostname)"
   ```

4. 运行以下命令，在 /etc/hosts 文件中为节点的主机名添加一个条目：

   ```
   $ echo "$IP $NAME" | sudo tee -a /etc/hosts >/dev/null
   ```

5. 通过在 /etc/microshift/config.yaml 中添加以下 YAML 片断来更新 MicroShift 配置文件：

   ```
   sudo tee /etc/microshift/config.yaml > /dev/null <<EOF
   node:
     hostnameOverride: hostnameOverride: $(echo $NAME)
   ```

```
  nodeIP: $(echo $IP)
EOF
```

6. MicroShift 现在可以使用回送设备进行集群通信。完成设备准备以离线使用。

   a. 如果设备当前附加了 NIC，请断开设备与网络的连接。

   b. 关闭该设备并断开 NIC。

   c. 重启该设备以使离线配置生效。

7. 运行以下命令重启 MicroShift 主机以应用配置更改：

```
$ sudo systemctl reboot 1
```

1 此步骤重启集群。在实施验证前，等待 greenboot 健康检查报告系统健康状态。

验证

此时，对 MicroShift 主机的网络访问已被严重。如果可以访问主机终端，您可以使用主机 CLI 验证集群是否已以稳定状态启动。

1. 输入以下命令验证 MicroShift 集群是否正在运行：

```
$ export KUBECONFIG=/var/lib/microshift/resources/kubeadmin/kubeconfig
$ sudo -E oc get pods -A
```

输出示例

```
NAMESPACE         NAME                                  READY   STATUS    RESTARTS
AGE
kube-system       csi-snapshot-controller-74d566564f-66n2f   1/1     Running   0
1m
kube-system       csi-snapshot-webhook-69bdff8879-xs6mb      1/1     Running   0
1m
```

```
openshift-dns                dns-default-dxglm                              2/2     Running   0          1m
openshift-dns                node-resolver-dbf5v                            1/1     Running   0          1m
openshift-ingress            router-default-8575d888d8-xmq9p                1/1     Running   0
1m
openshift-ovn-kubernetes     ovnkube-master-gcsx8                           4/4     Running   1
1m
openshift-ovn-kubernetes     ovnkube-node-757mf                             1/1     Running   1
1m
openshift-service-ca         service-ca-7d7c579f54-68jt4                    1/1     Running   0
1m
openshift-storage            topolvm-controller-6d777f795b-bx22r            5/5     Running   0
1m
openshift-storage            topolvm-node-fcf8l                             4/4     Running   0          1m
```