Red Hat build of MicroShift 4.17

ネットワーク

クラスターネットワークの設定および管理

Last Updated: 2024-12-04

Red Hat build of MicroShift 4.17 ネットワーク

クラスターネットワークの設定および管理

法律上の通知

概要

このドキュメントでは、DNS、ingress、Pod ネットワークなど、MicroShift クラスターネットワークを設定および管理する手順を説明します。

目次

第1章 OVN-KUBERNETES ネットワークプラグインについて

OVN-Kubernetes Container Network Interface (CNI) プラグインは、MicroShift クラスターのデフォルトのネットワークソリューションです。OVN-Kubernetes は、Open Virtual Network (OVN) に基づく Pod およびサービス用の仮想化ネットワークです。

- デフォルトのネットワーク設定と接続は、インストール中に **microshift-networking** RPM を使用して MicroShift に自動的に適用されます。
- OVN-Kubernetes ネットワークプラグインを使用するクラスターは、ノードで Open vSwitch (OVS) も実行します。
- OVN-K は、宣言されたネットワーク設定を実装するようにノードの OVS を設定します。
- ホストの物理インターフェイスは、デフォルトでは OVN-K ゲートウェイブリッジ **br-ex** にバインドされません。Network Manager CLI (**nmcli**) など、ホスト上の標準ツールを使用してデフォルトゲートウェイを管理できます。
- CNI の変更は MicroShift ではサポートされていません。

設定ファイルまたはカスタムスクリプトを使用して、次のネットワーク設定を設定できます。

- サブネットの CIDR 範囲を使用して、Pod に IP アドレスを割り当てることができます。
- 最大伝送単位 (MTU) 値を変更できます。
- ファイアウォールの Ingress と Egress を設定できます。
- MicroShift クラスターでは、Ingress ルールや Egress ルールなどのネットワークポリシーを定義できます。
- MicroShift Multus プラグインを使用して、他の CNI プラグインをチェーンできます。
- Ingress ルーターを設定または削除できます。

1.1. MICROSHIFT ネットワーク設定マトリックス

次の表はネットワーク機能のステータスをまとめたものです。デフォルトとして存在する機能、設定可能な機能、MicroShift サービスで使用できない機能があります。

表1.1 MicroShift ネットワーク機能の概要

ネットワーク機能	可用性	設定のサポート
アドレスのアドバタイズ	はい	はい [1]
Kubernetes ネットワークポリシー	はい	はい
Kubernetes ネットワークポリシーログ	利用不可	該当なし
負荷分散	はい	はい
マルチキャスト DNS	はい	はい [2]

ネットワーク機能	可用性	設定のサポート
ネットワークプロキシー	はい [3]	CRI-O
ネットワークパフォーマンス	はい	MTU 設定
Egress IP	利用不可	該当なし
Egress ファイアウォール	利用不可	該当なし
Egress ルーター	利用不可	該当なし
ファイアウォール	いいえ [4]	はい
ハードウェアのオフロード	利用不可	該当なし
ハイブリッドネットワーク	利用不可	該当なし
クラスター内通信の IPsec 暗号化	利用不可	該当なし
IPv6	サポート対象 [4]	該当なし
Ingress ルーター	はい	はい [6]
複数のネットワークプラグイン	はい	はい

1. 設定されていない場合、デフォルト値はサービスネットワークのすぐ後のサブネットに設定されます。たとえば、サービスネットワークが **10.43.0.0/16** の場合、**advertiseAddress** は、**10.44.0.0/32** に設定されます。

2. マルチキャスト DNS プロトコル (mDNS) を使用することで、**5353/UDP** ポートで公開されているマルチキャストを使用した、ローカルエリアネットワーク (LAN) 内で名前解決とサービス検出が可能になります。

3. MicroShift には、Egress トラフィックをプロキシーする、埋め込みの透過的機能はありません。Egress は手動で設定する必要があります。

4. firewalld サービスのセットアップは、RHEL for Edge でサポートされています。

5. IPv6 は、OVN-Kubernetes ネットワークプラグインを使用するシングルスタックおよびデュアルスタックネットワークの両方でサポートされています。IPv6 は、MicroShift Multus CNI プラグインを使用して他のネットワークに接続することでのみ使用できます。

6. MicroShift の **config.yaml** ファイルを使用して設定します。

1.1.1. デフォルトの設定

config.yaml ファイルを作成しない場合は、デフォルト値が使用されます。次の例は、デフォルトの設定を示しています。

- デフォルト値を確認するには、次のコマンドを実行します。

```
$ microshift show-config
```

YAML 形式でのデフォルト値の出力例

```
apiServer:
  advertiseAddress: 10.44.0.0/32 (1)
  auditLog:
    maxFileAge: 0
    maxFileSize: 200
    maxFiles: 10
    profile: Default
  namedCertificates:
    - certPath: ""
      keyPath: ""
      names:
        - ""
  subjectAltNames: []
debugging:
  logLevel: "Normal"
dns:
  baseDomain: microshift.example.com
etcd:
  memoryLimitMB: 0
ingress:
  listenAddress:
    - ""
  ports:
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed
  status: Managed
kubelet:
manifests:
  kustomizePaths:
    - /usr/lib/microshift/manifests
    - /usr/lib/microshift/manifests.d/*
    - /etc/microshift/manifests
    - /etc/microshift/manifests.d/*
network:
  clusterNetwork:
    - 10.42.0.0/16
  serviceNetwork:
    - 10.43.0.0/16
  serviceNodePortRange: 30000-32767
node:
  hostnameOverride: ""
  nodeIP: "" (2)
  nodeIPv6: ""
storage:
  driver: "" (3)
  optionalCsiComponents: (4)
    - ""
```

–

1. サービスネットワークのアドレスに基づいて計算されます。
2. デフォルトルートの IP アドレス。
3. デフォルトの null 値は、論理ボリュームマネージドストレージ(LVMS)をデプロイします。
4. デフォルト値は、**snapshot-controller** および **snapshot-webhook** をデプロイします。

1.2. ネットワーク機能

MicroShift 4.17 で利用できるネットワーク機能には、次のものがあります。

- Kubernetes ネットワークポリシー
- 動的ノード IP
- カスタムゲートウェイインターフェイス
- セカンドゲートウェイインターフェイス
- 指定されたホストインターフェイス上のクラスターネットワーク
- 特定のホストインターフェイス上の NodePort サービスへの外部アクセスをブロック

MicroShift 4.17 では利用できないネットワーク機能:

- Egress IP/ファイアウォール/QoS: 無効
- ハイブリッドネットワーク: サポートなし
- IPsec: サポートなし
- ハードウェアオフロード: サポートなし

1.3. IP 転送

ホストネットワーク **sysctl net.ipv4.ip_forward** カーネルパラメーターは、起動時に **ovnkube-master** コンテナーによって自動的に有効になります。これは、着信トラフィックを CNI に転送するために必要です。たとえば、**ip_forward** が無効になっている場合は、クラスターの外部から NodePort サービスにアクセスすると失敗します。

1.4. ネットワークパフォーマンスの最適化

デフォルトでは、リソース消費を最小限に抑えるために、OVS サービスに 3 つのパフォーマンス最適化が適用されます。

- **ovs-vswitchd.service** および **ovsdb-server.service** への CPU アフィニティー
- **no-mlockall** から **openvswitch.service**
- ハンドラーと **revalidator** のスレッドを **ovs-vswitchd.service** に限定

1.5. MICROSHIFT ネットワーキングコンポーネントおよびサービス

この簡単な概要では、MicroShift でのネットワークコンポーネントとその操作を説明します。**microshift-networking** RPM は、ネットワーク関連の依存関係と systemd サービスを自動的に取り込み、ネットワークを初期化するパッケージです (**microshift-ovs-init** systemd サービスなど)。

NetworkManager

MicroShift ノードで初期ゲートウェイブリッジをセットアップするには、NetworkManager が必要です。NetworkManager および **NetworkManager-ovs** RPM パッケージは、必要な設定ファイルを含む **microshift-networking** RPM パッケージへの依存関係としてインストールされます。MicroShift の NetworkManager は **keyfile** ファイルプラグインを使用し、**microshift-networking** RPM パッケージのインストール後に再起動されます。

microshift-ovs-init

microshift-ovs-init.service は、**microshift.service** に依存する systemd サービスとして、**microshift-networking** RPM パッケージによりインストールされます。OVS ゲートウェイブリッジを設定します。

OVN コンテナー

2 つの OVN-Kubernetes デーモンセットが MicroShift によってレンダリングおよび適用されます。

- **ovnkube-master** **northd**、**nbdb**、**sbdb**、および **ovnkube-master** コンテナーが含まれます。
- **ovnkube-node** ovnkube-node には、OVN-Controller コンテナーが含まれています。MicroShift の起動後、OVN-Kubernetes デーモンセットが **openshift-ovn-kubernetes** namespace にデプロイされます。

パッケージ

OVN-Kubernetes マニフェストと起動ロジックは MicroShift に組み込まれています。**microshift-networking** RPM に含まれる systemd サービスと設定は次のとおりです。

- **NetworkManager.service** の **/etc/NetworkManager/conf.d/microshift-nm.conf**
- **ovs-vswitchd.service** の **/etc/systemd/system/ovs-vswitchd.service.d/microshift-cpuaffinity.conf**
- **ovs-server.service** の **/etc/systemd/system/ovsdb-server.service.d/microshift-cpuaffinity.conf**
- **microshift-ovs-init.service** の **/usr/bin/configure-ovs-microshift.sh**
- **microshift-ovs-init.service** の **/usr/bin/configure-ovs.sh**
- CRI-O サービスの **/etc/crio/crio.conf.d/microshift-ovn.conf**

1.6. ブリッジマッピング

ブリッジマッピングにより、プロバイダーネットワークのトラフィックは、物理ネットワークに到達することが可能となります。トラフィックはプロバイダーネットワークから出て、**br-int** ブリッジに到達します。**br-int** と **br-ex** の間のパッチポートは、トラフィックがプロバイダーネットワークとエッジネットワークを通信できるようにします。Kubernetes Pod は、仮想イーサネットペアを介して **br-int** ブリッジに接続されます。仮想イーサネットペアの一端は Pod の namespace に接続され、他端は **br-int** ブリッジに接続されます。

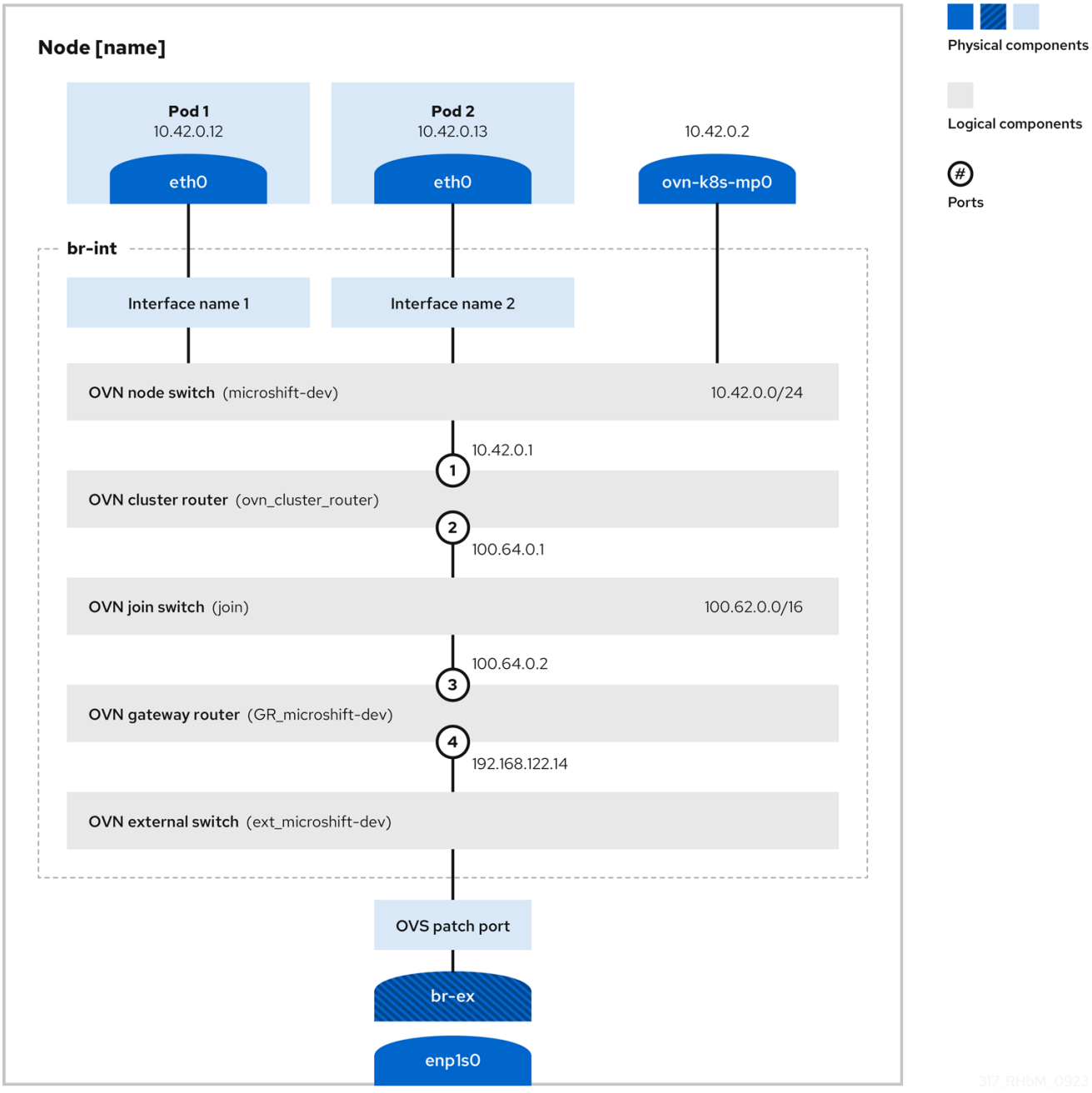
1.7. ネットワークトポロジー

OVN-Kubernetes は、オーバーレイベースのネットワーク実装を提供します。このオーバーレイには、Service および NetworkPolicy の OVS ベースの実装が含まれています。オーバーレイネットワークは、Geneve (Generic Network Virtualization Encapsulation) トンネルプロトコルを使用します。Geneve トンネルの Pod 最大伝送単位 (MTU) が設定されていない場合、デフォルトルートの MTU に設定されます。

MTU を設定する際には、ホスト上の物理インターフェイスの MTU 以下の値を設定する必要があります。その MTU 以下の値を設定することで、送信前にトンネルヘッダーに追加される必要な情報のための容量が確保されます。

OVS は MicroShift ノードで systemd サービスとして実行します。OVS RPM パッケージは、**microshift-networking** RPM パッケージへの依存関係としてインストールされます。OVS は、**microshift-networking** RPM がインストールされるとすぐに開始します。

Red Hat build of MicroShift ネットワークトポロジー



1.7.1. 仮想化ネットワークの OVN 論理コンポーネントの説明

OVN ノードスイッチ

<node-name> という名前の仮想スイッチ。OVN ノードスイッチの名前は、ノードのホスト名に基づいて付けられます。

- この例では、**node-name** は **microshift-dev** です。

OVN クラスタールーター

ovn_cluster_router という名前の仮想ルーター。分散ルーターとも呼ばれます。

- この例では、クラスターネットワークは **10.42.0.0/16** です。

OVN 結合スイッチ

join という名前の仮想スイッチ。

OVN ゲートウェイルーター

GR_<node-name> という名前の仮想ルーター。外部ゲートウェイルーターとも呼ばれます。

OVN 外部スイッチ

ext_<node-name> という名前の仮想スイッチ。

1.7.2. ネットワークトポロジー図の接続の説明

- ネットワークサービスと OVN 外部スイッチ **ext_microshift-dev** の間の North-South トラフィックは、ゲートウェイブリッジ **br-ex** によってホストカーネルを介して提供されます。
- OVN ゲートウェイルーター **GR_microshift-dev** は、論理ルーターポート 4 を介して外部ネットワークスイッチ **ext_microshift-dev** に接続しています。ポート 4 にはノード IP アドレス 192.168.122.14 が割り当てられます。
- 参加スイッチ **join** は、OVN ゲートウェイルーター **GR_microshift-dev** を OVN クラスタールーター **ovn_cluster_router** に接続します。IP アドレス範囲は 100.62.0.0/16 です。
  - OVN ゲートウェイルーター **GR_microshift-dev** は、論理ルーターポート 3 を介して OVN 結合スイッチ **join** に接続します。ポート 3 は内部 IP アドレス 100.64.0.2 に接続します。
  - OVN クラスタールーター **ovn_cluster_router** は、論理ルーターポート 2 を介して参加スイッチ **join** に接続します。ポート 2 は内部 IP アドレス 100.64.0.1 に接続します。
- OVN クラスタールーター **ovn_cluster_router** は、論理ルーターポート 1 を介してノードスイッチ **microshift-dev** に接続します。ポート 1 には、OVN クラスターネットワーク IP アドレス 10.42.0.1 が割り当てられます。
- Pod とネットワークサービス間の East-West トラフィックは、OVN クラスタールーター **ovn_cluster_router** とノードスイッチ **microshift-dev** によって提供されます。IP アドレス範囲は 10.42.0.0/24 です。
- Pod 間の East-West トラフィックは、ネットワークアドレス変換 (NAT) を使用せずに、ノードスイッチ **microshift-dev** によって提供されます。
- Pod と外部ネットワーク間の North-South トラフィックは、OVN クラスタールーター **ovn_cluster_router** とホストネットワークによって提供されます。このルーターは、**ovn-kubernetes** 管理ポート **ovn-k8s-mp0** を介して、IP アドレス 10.42.0.2 で接続しています。
- すべての Pod は、インターフェイスを介して OVN ノードスイッチに接続します。

- この例では、Pod 1 と Pod 2 は、**Interface 1** と **Interface 2** を介してノードスイッチに接続しています。

1.8. 関連情報

- YAML 設定ファイルの使用
- ネットワーク設定について
- 複数ネットワークの使用について
- ネットワークポリシーについて

第2章 ネットワーク設定について

ネットワークのカスタマイズとデフォルト設定を MicroShift デプロイメントに適用する方法を学びます。各ノードは単一のマシンと単一の MicroShift に含まれているため、デプロイごとに個別の構成、Pod、および設定が必要です。

クラスター管理者は、クラスターで実行されるアプリケーションを外部トラフィックに公開し、ネットワーク接続のセキュリティーを保護するための複数のオプションがあります。

- NodePort などのサービス
- **Ingress** や **Route** などの API リソース

デフォルトで、Kubernetes は各 Pod に、Pod 内で実行しているアプリケーションの内部 IP アドレスを割り当てます。Pod とそのコンテナーの間にはトラフィックを配置できますが、NodePort などのサービスで公開されている場合を除き、クラスター外のクライアントは Pod に直接ネットワークアクセスできません。

2.1. OVN-KUBERNETES 設定ファイルの作成

OVN-Kubernetes 設定ファイルが作成されていない場合、MicroShift は組み込みのデフォルト OVN-Kubernetes 値を使用します。OVN-Kubernetes 設定ファイルを **/etc/microshift/ovn.yaml** に書き込むことができます。設定用のサンプルファイルが提供されています。

手順

1. **ovn.yaml** ファイルを作成するには、次のコマンドを実行します。

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

2. 作成した設定ファイルの内容を一覧表示するには、次のコマンドを実行します。

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

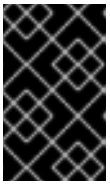
デフォルトの最大伝送単位 (MTU) 値を含む YAML ファイルの例

```
mtu: 1400
```

3. 設定をカスタマイズするには、MTU 値を変更します。以下の表で詳細を記載します。

表2.1 サポート対象の MicroShift の OVN-Kubernetes 設定 (任意)

フィールド	タイプ	デフォルト	説明	例
mtu	uint32	auto	Pod に使用される MTU 値	1300



重要

ovn.yaml ファイルの **mtu** 設定値を変更した場合に、更新された設定を適用するには、Red Hat build of MicroShift が実行しているホストを再起動する必要があります。

カスタム ovn.yaml 設定ファイルの例

```
mtu: 1300
```

2.2. OVNKUBE-MASTER POD の再起動

次の手順で **ovnkube-master** Pod を再起動します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- インフラストラクチャーにインストールされたクラスターが OVN-Kubernetes ネットワークプラグインで設定されている。
- KUBECONFIG 環境変数が設定されている。

手順

次の手順を使用して、**ovnkube-master** Pod を再起動します。

1. 次のコマンドを実行して、リモートクラスターにアクセスします。

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

2. 次のコマンドを実行して、再起動する **ovnkube-master** Pod の名前を見つけます。

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

3. 次のコマンドを実行して、**ovnkube-master** Pod を削除します。

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

4. 次のコマンドを使用して、新しい **ovnkube-master** Pod が実行されていることを確認します。

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

実行中の Pod のリストには、新しい **ovnkube-master** Pod の名前と経過時間が表示されます。

2.3. HTTP または HTTPS プロキシーの背後への MICROSHIFT のデプロイ

基本的な匿名性とセキュリティー対策を Pod に追加する場合は、HTTP または HTTPS プロキシーの背後に MicroShift クラスターをデプロイします。

プロキシーの背後に MicroShift をデプロイする場合は、HTTP または HTTPS 要求を開始するすべてのコンポーネントでプロキシーサービスを使用するように、ホストオペレーティングシステムを設定する必要があります。

クラウドサービスへのアクセスなど、Egress トラフィックを伴うユーザー固有のワークロードまたは Pod はすべて、プロキシーを使用するように設定する必要があります。MicroShift には、Egress トラフィックをプロキシーする、埋め込みの透過的機能はありません。

2.4. RPM-OSTREE HTTP または HTTPS プロキシーの使用

RPM-OStree で HTTP または HTTPS プロキシーを使用するには、設定ファイルに **Service** セクションを追加し、**rpm-ostree** サービスの **http_proxy** 環境変数を設定する必要があります。

手順

1. 次の設定を **/etc/systemd/system/rpm-ostreed.service.d/00-proxy.conf** ファイルに追加します。

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

2. 次に、設定をリロードして、サービスを再起動して変更を適用します。

   a. 次のコマンドを実行して、設定をリロードします。

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

   b. 次のコマンドを実行して、**rpm-ostreed** サービスを再起動します。

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

2.5. CRI-O コンテナーランタイムでのプロキシーの使用

CRI-O で HTTP または HTTPS プロキシーを使用するには、設定ファイルに **Service** セクションを追加し、**HTTP_PROXY** および **HTTPS_PROXY** 環境変数を設定する必要があります。**NO_PROXY** 変数を設定して、ホストのリストをプロキシーから除外することもできます。

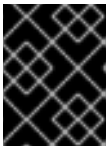
手順

1. 設定ファイル用のディレクトリーが存在しない場合は、作成します。

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

2. 次の設定を **/etc/systemd/system/crio.service.d/00-proxy.conf** ファイルに追加します。

```
[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

環境変数の設定ファイルの **Service** セクションを定義する必要があります。定義しないと、プロキシー設定が適用されません。

3. 設定を再読み込みします。

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

4. CRI-O サービスを再起動します。

```
$ sudo systemctl restart crio
```

5. MicroShift サービスを再起動して設定を適用します。

```
$ sudo systemctl restart microshift
```

検証

1. 次のコマンドを実行し、出力を調べて、Pod が起動していることを確認します。

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

2. 次のコマンドを実行し、出力を調べて、MicroShift がコンテナーイメージをプルできることを確認します。

```
$ sudo crictl images
```

2.6. 実行中のクラスターからの OVS インターフェイスのスナップショット取得

スナップショットは、特定の時点における OVS インターフェイスの状態とデータを表します。

手順

- 実行中の MicroShift クラスターから OVS インターフェイスのスナップショットを表示するには、次のコマンドを使用します。

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

実行中のクラスターでの OVS インターフェイスの例

```
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** と **patch-br-int-to-br-ex_localhost.localdomain** は、**br-ex** と **br-int** を接続する OVS パッチポートです。
2. **patch-br-ex_localhost.localdomain-to-br-int** と **patch-br-int-to-br-ex_localhost.localdomain** は、**br-ex** と **br-int** を接続する OVS パッチポートです。
3. Pod インターフェイス **eebee1ce5568761** は、Pod Sandbox ID の最初の 15 ビットを使用して名前が付けられ、**br-int** ブリッジにプラグインされます。
4. Pod インターフェイス **b47b1995ada84f4** は、Pod Sandbox ID の最初の 15 ビットを使用して名前が付けられ、**br-int** ブリッジにプラグインされます。
5. Pod インターフェイス **3031f43d67c167f** は、Pod Sandbox ID の最初の 15 ビットを使用して名前が付けられ、**br-int** ブリッジにプラグインされます。
6. ヘアピントラフィック用の OVS 内部ポート **ovn-k8s-mp0** は、**ovnkube-master** コンテナーによって作成されます。

2.7. ワークロード用の MICROSHIFT LOADBALANCER サービス

MicroShift には、クラスター内のワークロードとアプリケーションに使用できる Network Load Balancer の実装が組み込まれています。Ingress ルールを理解し、Ingress コントローラーとして機能するように Pod を設定することで、**LoadBalancer** サービスを作成できます。以下の手順では、デプロイメントベースの **LoadBalancer** サービスの例を示します。

2.8. アプリケーション用のロードバランサーのデプロイ

次の手順例では、ノード IP アドレスを **LoadBalancer** サービス設定ファイルの外部 IP アドレスとして使用します。この例を、ロードバランサーをデプロイする方法のガイダンスとして使用します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- OVN-Kubernetes ネットワークプラグインで設定されたインフラストラクチャーにクラスターがインストールされている。
- **KUBECONFIG** 環境変数が設定されている。

手順

1. 次のコマンドを入力して、Pod が実行されていることを確認します。

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

出力例

```
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. 次のコマンドを実行して、namespace を作成します。

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

1 _<nginx-lb-test> は、作成するアプリケーション namespace に置き換えます。

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

namespace の例

次の例では、**nginx** テストアプリケーションの 3 つのレプリカを、作成された namespace にデプロイします。

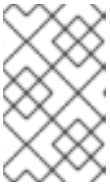
```
oc apply -n $NAMESPACE -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx
data:
  headers.conf: |
    add_header X-Server-IP  \$server_addr always;
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    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. 次のコマンドを実行すると、3 つのサンプルレプリカが正常に開始したことを確認できます。

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

4. 次のコマンドを実行して、**nginx** テストアプリケーションの **LoadBalancer** サービスを作成します。

```
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
```



注記

port パラメーターが、他の **LoadBalancer** サービスまたは MicroShift コンポーネントによって占有されていないホストポートであることを確認する必要があります。

5. サービスファイルが存在し、外部 IP アドレスが適切に割り当てられていること、および外部 IP がノード IP と同一であることを確認するには、次のコマンドを実行します。

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

出力例

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

検証

次のコマンドは、**LoadBalancer** サービス設定の外部 IP アドレスを使用して、例の **nginx** アプリケーションへの 5 つの接続を形成します。コマンドの結果は、それらのサーバー IP アドレスのリストです。

- 次のコマンドを実行して、ロードバランサーが実行中のすべてのアプリケーションにリクエストを送信していることを確認します。

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

LoadBalancer サービスがトラフィックをアプリケーションに正常に分散している場合、前のコマンドの出力には異なる IP アドレスが含まれます。次に例を示します。

出力例

```
-
```

```
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. 特定のホストインターフェイス上の NODEPORT サービスへの外部アクセスをブロックする

OVN-Kubernetes は、Red Hat build of MicroShift ノードの外部から NodePort サービスにアクセスできるホストインターフェイスを制限しません。次の手順では、特定のホストインターフェイスで NodePort サービスをブロックし、外部アクセスを制限する方法を説明します。

前提条件

- root 権限を持つアカウント。

手順

1. 次のコマンドを実行して、**NODEPORT** 変数を Kubernetes NodePort サービスに割り当てられたホストポート番号に変更します。

```
# export NODEPORT=30700
```

2. **INTERFACE_IP** 値を、ブロックするホストインターフェイスの IP アドレスに変更します。以下に例を示します。

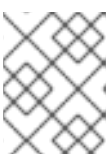
```
# export INTERFACE_IP=192.168.150.33
```

3. **nat** テーブル PREROUTING チェーンに新しいルールを挿入して、宛先ポートと IP アドレスに一致するすべてのパケットをドロップします。以下に例を示します。

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

4. 次のコマンドを実行して、新しいルールをリスト表示します。

```
$ 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
 }
}
```



注記

新しく追加したルールの **handle** 番号をメモします。次の手順で **handle** 番号を削除する必要があります

5. 次のサンプルコマンドを使用してカスタムルールを削除します。

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

2.10. マルチキャスト DNS プロトコル

マルチキャスト DNS プロトコル (mDNS) を使用することで、**5353/UDP** ポートで公開されているマルチキャストを使用した、ローカルエリアネットワーク (LAN) 内で名前解決とサービス検出が可能になります。

MicroShift には、権威 DNS サーバーを MicroShift のサービスに対してクライアントを参照するように再設定できないデプロイメントシナリオ向けに、埋め込みの mDNS サーバーが含まれています。埋め込みの DNS サーバーにより、MicroShift によって公開された **.local** ドメインが LAN 上の他の要素によって検出されるようになります。

2.11. 公開されたネットワークポートの監査

MicroShift では、次の場合にワークロードによってホストポートを開くことができます。ログを確認してネットワークサービスを表示できます。

2.11.1. hostNetwork

Pod が **hostNetwork:true** で設定されている場合、Pod はホストネットワーク namespace で実行されています。この設定では、ホストポートを独立して開くことができます。MicroShift コンポーネントログを使用してこのケースを追跡することはできません。ポートは firewalld ルールの対象となります。firewalld でポートが開いている場合は、firewalld デバッグログでポートの開きを確認できます。

前提条件

- ビルドホストへの root ユーザーアクセス権がある。

手順

1. オプション: 次のコマンド例を使用して、**hostNetwork:true** パラメーターが ovnkube-node Pod に設定されていることを確認できます。

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

2. 次のコマンドを実行して、firewalld ログのデバッグを有効にします。

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

3. firewalld サービスを再起動します。

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

4. デバッグオプションが適切に追加されたことを確認するには、次のコマンドを実行します。

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

firewalld デバッグログは、**/var/log/firewalld** パスに保存されます。

ポートオープンルールが追加されたときのログの例:

```
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')
```

ポートオープンルールが削除されたときのログの例:

```
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

MicroShift で hostPort 設定ログにアクセスできます。次のログは、hostPort 設定の例です。

手順

- 次のコマンドを実行すると、ログにアクセスできます。

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

ホストポートが開いている場合の CRI-O ログの例:

```
$ 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"
```

ホストポートが閉じている場合の CRI-O ログの例:

```
$ 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 および LoadBalancer サービス

OVN-Kubernetes は、**NodePort** および **LoadBalancer** サービスタイプのホストポートを開きます。これらのサービスは、ホストポートから Ingress トラフィックを取得し、それを clusterIP に転送する iptables ルールを追加します。**NodePort** および **LoadBalancer** サービスのログを次の例に示します。

手順

1. **ovnkube-master** Pod の名前にアクセスするには、次のコマンドを実行します。

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

ovnkube-master Pod 名の例

```
ovnkube-master-n2shv
```

2. **ovnkube-master** Pod を使用し、次のコマンド例を実行すると、 **NodePort** および **LoadBalancer** サービスのログにアクセスできます。

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

NodePort サービス:

ホストポートが開いている場合の ovnkube-master Pod の ovnkube-master コンテナー内のログの例:

```
$ 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
```

ホストポートが閉じている場合の ovnkube-master Pod の ovnkube-master コンテナー内のログの例:

```
$ 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 サービス:

ホストポートが開いている場合の ovnkube-master Pod の ovnkube-master コンテナー内のログの例:

```
$ 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
```

ホストポートが閉じている場合の ovnkube-master Pod の ovnkube-master コンテナー内のログの例:

```
$ 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 ルーターの概要および設定

MicroShift を使用してルーターとルート許可ポリシーを設定するためのデフォルト設定とカスタム設定について学習します。

3.1. ルーターの設定について

Ingress をオプションにするには、MicroShift Ingress ルーター設定を指定して、ネットワークトラフィックに公開されるポート (存在する場合) を管理できます。指定されたルーティングは、Ingress ロードバランシングの例です。

- デフォルトの Ingress ルーターは常にオンになっており、**http: 80** および **https: 443** ポート上のすべての IP アドレスで実行されます。
- デフォルトのルーター設定では、任意の namespace にアクセスできます。

MicroShift 上で実行されるアプリケーションによっては、デフォルトのルーターは必要なく、代わりに独自のルーターが作成される場合があります。ルーターを設定して、Ingress と namespace アクセスの両方を制御できます。

ヒント

設定を開始する前に **oc get deployment -n openshift-ingress** コマンドを使用して MicroShift インストールにデフォルトルーターが存在するかどうかを確認できます。このコマンドは、次の出力を返します。

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

3.1.1. ルーターの設定および有効な値

Ingress ルーターの設定は、以下のパラメーターおよび有効な値で構成されます。

config.yaml ルーター設定の例

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

(1) **ingress.listenAddress** の値は、デフォルトでホストのネットワーク全体に設定されます。有効なカスタマイズ可能な値は、単一の IP アドレスまたはホスト名、あるいは IP アドレスまたはホスト名のリストです。

(2) 両方のポートエントリーの有効な値は、1 - 65535 の範囲にある単一の一意のポートです。**ports.http** および **ports.https** フィールドの値は、同じにすることはできません。

3 デフォルト値。ルートが、複数の namespace において名前が同じでパスが異なる要求を許可します。

4 デフォルト値。Ingress ポートを開いたままにするには、**Managed** が必要です。

重要

デフォルトの MicroShift ルーターとルーターを有効化二設定することで、firewalld サービスはバイパスされます。ルーターがアクティブな場合にネットワークポリシーを設定し、Ingress および Egress を制御する必要があります。

3.2. ルーターの無効化

MicroShift Pod がサウスバウンドの運用システムとノースバウンドのクラウドデータシステムのみに接続する必要がある産業用 IoT スペースなどのユースケースでは、インバウンドサービスは必要ありません。このような Egress のみのユースケースでルーターを無効にするには、この手順を使用します。

前提条件

- MicroShift をインストールしている。
- MicroShift の **config.yaml** ファイルが作成されている。
- OpenShift CLI (**oc**) がインストールされている。

ヒント

MicroShift **config.yaml** ファイルで指定する必要のある設定をすべて同時に完了すると、システムの再起動を最小限に抑えることができます。

手順

1. 次の例に示すように、MicroShift **config.yaml** ファイルの **ingress.status** フィールドの値を **Removed** に更新します。

   config.yaml Ingress スタンザの例

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

   1 値が **Removed** に設定されている場合、**ingress.ports** にリストされているポートは自動的に閉じられます。**Ingress** スタンザ内のその他の設定 (**routeAdmissionPolicy.namespaceOwnership** フィールドの値など) は無視されます。

2. 次のコマンドを実行して、MicroShift サービスを再起動します。

```
$ sudo systemctl restart microshift
```

注記

MicroShift サービスは、再起動時に現在の設定を出力します。

検証

- システムが再起動されると、次のコマンドを実行して、ルーターが削除され、Ingress が停止されていることを確認します。

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

予想される出力

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

3.3. ルーター INGRESS の設定

MicroShift アプリケーションがデータトラフィックのみをリッスンする必要がある場合は、**listenAddress** 設定を指定してデバイスを分離できます。ネットワーク接続用の特定のポートと IP アドレスを設定することもできます。ユースケースに合わせてエンドポイント設定をカスタマイズするために必要な組み合わせを使用します。

3.3.1. ルーターポートの設定

ルーター Ingress フィールドを設定することで、デバイスが使用するポートを制御できます。

前提条件

- MicroShift をインストールしている。
- MicroShift の **config.yaml** ファイルが作成されている。
- OpenShift CLI (**oc**) がインストールされている。

ヒント

MicroShift **config.yaml** ファイルで指定する必要のある設定をすべて同時に完了すると、システムの再起動を最小限に抑えることができます。

手順

1. **ingress.ports.http** および **ingress.ports.https** フィールドの MicroShift **config.yaml** ポート値を、使用するポートに更新します。

config.yaml ルーター設定の例

```
# ...
ingress:
  ports: 1
    http: 80
```

```
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed
  status: Managed 2
# ...
```

1 デフォルトのポートが表示されます。カスタマイズ可能です。両方のポートエントリーの有効な値は、1 - 65535 の範囲にある単一の一意のポートです。**ports.http** および **ports.https** フィールドの値は、同じにすることはできません。

2 デフォルトの値です。Ingress ポートを開いたままにするには、**Managed** が必要です。

2. 次のコマンドを実行して、MicroShift サービスを再起動します。

```
$ sudo systemctl restart microshift
```

3.3.2. ルーターの IP アドレスの設定

特定の IP アドレスを設定することで、ルーターへのネットワークトラフィックを制限できます。以下に例を示します。

- ルーターが内部ネットワークでのみアクセス可能で、ノースバウンドのパブリックネットワークではアクセスできないユースケース
- ルーターがノースバウンドのパブリックネットワークからのみアクセス可能で、内部ネットワークからはアクセスできないユースケース
- ルーターが内部ネットワークとノースバウンドのパブリックネットワークの両方から到達可能であるが、別々の IP アドレス上にある場合のユースケース

前提条件

- MicroShift をインストールしている。
- MicroShift の **config.yaml** ファイルが作成されている。
- OpenShift CLI (**oc**) がインストールされている。

ヒント

MicroShift **config.yaml** ファイルで指定する必要のある設定をすべて同時に完了すると、システムの再起動を最小限に抑えることができます。

手順

1. 要件に応じて、次の例に示すように、MicroShift **config.yaml** の **ingress.listenAddress** フィールドのリストを更新します。

   デフォルトのルーター IP アドレスリスト

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

```
    - "<host_network>" (1)
# ...
```

(1) **ingress.listenAddress** の値は、デフォルトでホストのネットワーク全体に設定されます。デフォルトのリストを引き続き使用するには、MicroShift **config.yaml** ファイルから **listen.Address** フィールドを削除します。このパラメーターをカスタマイズするには、リストを使用します。リストには、単一の IP アドレスまたは NIC 名、あるいは複数の IP アドレスと NIC 名を含めることができます。

重要

config.yaml ファイルを使用する場合は、**listenAddress** パラメーターを削除するか、リスト形式で値を追加する必要があります。フィールドを空白のままにしないでください。空白のままにすると、再起動時に MicroShift がクラッシュします。

単一のホスト IP アドレスを持つルーターの設定例

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

IP アドレスと NIC 名を組み合わせたルーター設定の例

```
# ...
ingress:
  listenAddress:
    - 10.2.1.100
    - 10.2.2.10
    - ens3
# ...
```

2. 次のコマンドを実行して、MicroShift サービスを再起動します。

```
$ sudo systemctl restart microshift
```

検証

- 設定が適用されていることを確認するには、**ingress.listenAddress** の IP アドレスに到達可能であることを確認してから、これらのロードバランサー IP アドレスのいずれかを宛先とするルートを **curl** で取得します。

3.4. 関連情報

- デフォルト設定 (MicroShift)
- ネットワークポリシーについて

3.5. ルートの受付ポリシーの設定

デフォルトでは、MicroShift は複数の namespace 内のルートが同じホスト名を使用することを許可します。ルートのポリシーを設定することで、namespace が異なる同じホスト名をルートが要求しないようにできます。

前提条件

- MicroShift をインストールしている。
- MicroShift の **config.yaml** ファイルが作成されている。
- OpenShift CLI (**oc**) がインストールされている。

ヒント

MicroShift **config.yaml** ファイルで指定する必要のある設定をすべて同時に完了すると、システムの再起動を最小限に抑えることができます。

手順

1. 異なる namespace 内のルートが同じホスト名を要求しないようにするには、MicroShift **config.yaml** ファイルで **namespaceOwnership** フィールドの値を **Strict** に更新します。以下の例を参照してください。

config.yaml ルート受付ポリシーの例

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

1 異なる namespace のルートが同じホストを要求しないようにします。有効な値は **Strict** および **InterNamespaceAllowed** です。カスタマイズされた **config.yaml** 内の値を削除すると、**InterNamespaceAllowed** 値が自動的に設定されます。

2. 設定を適用するには、次のコマンドを実行して MicroShift サービスを再起動します。

```
$ sudo systemctl restart microshift
```

第4章 ネットワークポリシー

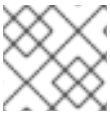
4.1. ネットワークポリシーについて

MicroShift でネットワークポリシーがどのように機能し、クラスター内の Pod へのネットワークトラフィックを制限または許可するかを説明します。

4.1.1. MicroShift でのネットワークポリシーの仕組み

MicroShift のデフォルトの OVN-Kubernetes Container Network Interface (CNI) プラグインを使用するクラスターでは、ネットワーク分離は、ホスト上で設定されている firewalld と MicroShift 内で作成された **NetworkPolicy** オブジェクトの両方によって制御されます。firewalld と **NetworkPolicy** の同時使用がサポートされています。

- ネットワークポリシーは、OVN-Kubernetes によって制御されるトラフィックの境界内でのみ機能するため、**hostPort/hostNetwork** が有効な Pod を除くあらゆる状況に適用できます。
- また、firewalld 設定は、**hostPort/hostNetwork** が有効な Pod には適用されません。



注記

firewalld ルールは、**NetworkPolicy** が適用される前に実行されます。

警告

ネットワークポリシーは、ホストのネットワーク namespace には適用されません。ホストネットワークが有効にされている Pod はネットワークポリシールールによる影響を受けません。ただし、ホストネットワーク化された Pod に接続する Pod はネットワークポリシールールの影響を受ける可能性があります。

ネットワークポリシーではローカルホストからのトラフィックをブロックできません。

デフォルトでは、MicroShift ノード内のすべての Pod は、他の Pod およびネットワークエンドポイントからアクセスできます。クラスターで 1 つ以上の Pod を分離するには、許可される受信接続を示す **NetworkPolicy** オブジェクトを作成してください。**NetworkPolicy** オブジェクトを作成および削除できます。

Pod が 1 つ以上の **NetworkPolicy** オブジェクトのセレクターと一致する場合、Pod はそれらの **NetworkPolicy** オブジェクトの少なくとも 1 つにより許可された接続だけを使用できます。**NetworkPolicy** オブジェクトによって選択されていない Pod は完全にアクセス可能です。

ネットワークポリシーは、TCP、UDP、ICMP、および SCTP プロトコルにのみ適用されます。他のプロトコルは影響を受けません。

以下のサンプル **NetworkPolicy** オブジェクトは、複数の異なるシナリオをサポートすることを示しています。

- すべてのトラフィックを拒否します。

プロジェクトに deny by default (デフォルトで拒否) を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

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

- MicroShift の Ingress であるデフォルトルーターからの接続を許可します。
  MicroShift デフォルトルーターからの接続を許可するには、次の **NetworkPolicy** オブジェクトを追加します。

```
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
```

- 同じ namespace 内の Pod からの接続のみを受け入れます。
  Pod が同じ namespace 内の他の Pod からの接続を受け入れるが、他の namespace 内の Pod からの接続はすべて拒否するようにするには、次の **NetworkPolicy** オブジェクトを追加します。

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

- Pod ラベルに基づいて HTTP および HTTPS トラフィックのみを許可します。
  特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```
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
```

- namespace および Pod セレクターの両方を使用して接続を受け入れます。
namespace と Pod セレクターを組み合わせてネットワークトラフィックのマッチングをするには、以下と同様の **NetworkPolicy** オブジェクトを使用できます。

```
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 オブジェクトは加算されるものです。つまり、複数の **NetworkPolicy** オブジェクトを組み合わせて複雑なネットワーク要件を満すことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、**allow-same-namespace** および **allow-http-and-https** ポリシーの両方を定義できます。この設定により、**role=frontend** ラベルの付いた Pod は各ポリシーで許可されている接続をすべて受け入れることができます。つまり、同じ namespace の Pod からのすべてのポート、およびすべての namespace の Pod からのポート **80** および **443** での接続を受け入れます。

4.1.2. OVN-Kubernetes ネットワークプラグインによるネットワークポリシーの最適化

ネットワークポリシーを設計する場合は、以下のガイドラインを参照してください。

- 同じ **spec.podSelector** 仕様を持つネットワークポリシーの場合、**ingress** ルールまたは **egress** ルールを持つ複数のネットワークポリシーを使用するよりも、複数の **Ingress** ルールまたは **egress** ルールを持つ 1 つのネットワークポリシーを使用する方が効率的です。

- **podSelector** または **namespaceSelector** 仕様に基づくすべての **Ingress** または **egress** ルールは、**number of pods selected by network policy + number of pods selected by ingress or egress rule** に比例する数の OVS フローを生成します。そのため、Pod ごとに個別のルールを作成するのではなく、1 つのルールで必要な数の Pod を選択できる **podSelector** または **namespaceSelector** 仕様を使用することが推奨されます。
たとえば、以下のポリシーには 2 つのルールが含まれています。

```
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
```

以下のポリシーは、上記と同じ 2 つのルールを 1 つのルールとして表現しています。

```
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]}
```

同じガイドラインが **spec.podSelector** 仕様に適用されます。異なるネットワークポリシーに同じ **ingress** ルールまたは **egress** ルールがある場合、共通の **spec.podSelector** 仕様で 1 つのネットワークポリシーを作成する方が効率的な場合があります。たとえば、以下の 2 つのポリシーには異なるルールがあります。

```
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
```

以下のネットワークポリシーは、上記と同じ 2 つのルールを 1 つのルールとして表現しています。

```
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
```

この最適化は、複数のセレクターを 1 つのセレクターとして表現する場合に限り適用できます。セレクターが異なるラベルに基づいている場合、この最適化は適用できない可能性があります。その場合は、ネットワークポリシーの最適化に特化して新規ラベルをいくつか適用することを検討してください。

4.2. ネットワークポリシーの作成

namespace のネットワークポリシーを作成できます。

4.2.1. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
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 オブジェクトの名前。
2. ポリシーが適用される Pod を説明するセレクター。
3. ポリシーオブジェクトが入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namaspace にある Pod を照合して検索します。
4. トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

4.2.2. CLI を使用したネットワークポリシーの作成

クラスターの namespace に許可される Ingress または Egress ネットワークトラフィックを記述する詳細なルールを定義するには、ネットワークポリシーを作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. ポリシールールを作成します。
   a. **<policy_name>.yaml** ファイルを作成します。

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

   ここでは、以下のようになります。

   <policy_name>
   ネットワークポリシーファイル名を指定します。

   b. 作成したばかりのファイルで、以下の例のようなネットワークポリシーを定義します。

   すべての namespace のすべての Pod から Ingress を拒否します。

   これは基本的なポリシーであり、他のネットワークポリシーの設定によって許可されたクロス Pod トラフィック以外のすべてのクロス Pod ネットワーキングをブロックします。

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

   同じ namespace のすべての Pod から Ingress を許可します。

```
kind: NetworkPolicy
```

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

特定の namespace から 1つの Pod への上りトラフィックを許可する

このポリシーは、**namespace-y** で実行されている Pod から **pod-a** というラベルの付いた Pod へのトラフィックを許可します。

```
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. ネットワークポリシーオブジェクトを作成するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーファイル名を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

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

4.2.3. デフォルトの全拒否ネットワークポリシーの作成

これは基本的なポリシーであり、他のデプロイメントされたネットワークポリシーの設定によって許可されたネットワークトラフィック以外のすべてのクロス Pod ネットワークをブロックします。この手順では、デフォルトの **deny-by-default** ポリシーを適用します。

前提条件

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. すべての namespace におけるすべての Pod からの Ingress を拒否する **deny-by-default** ポリシーを定義する次の YAML を作成します。YAML を **deny-by-default.yaml** ファイルに保存します。

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

(1) **namespace: default** は、このポリシーを **default** namespace にデプロイします。

(2) **podSelector:** は空です。これは、すべての Pod に一致することを意味します。したがって、ポリシーはデフォルト namespace のすべての Pod に適用されます。

(3) 指定された **ingress** ルールはありません。これにより、着信トラフィックがすべての Pod にドロップされます。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

4.2.4. 外部クライアントからのトラフィックを許可するネットワークポリシーの作成

deny-by-default ポリシーを設定すると、外部クライアントからラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーの設定に進むことができます。

注記

firewalld ルールは、**NetworkPolicy** が適用される前に実行されます。

この手順に従って、パブリックインターネットから直接、またはロードバランサーを使用して Pod にアクセスすることにより、外部サービスを許可するポリシーを設定します。トラフィックは、ラベル **app=web** を持つ Pod にのみ許可されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. パブリックインターネットからのトラフィックが直接、またはロードバランサーを使用して Pod にアクセスできるようにするポリシーを作成します。YAML を **web-allow-external.yaml** ファイルに保存します。

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

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

4.2.5. すべての namespace からアプリケーションへのトラフィックを許可するネットワークポリシーを作成する

この手順に従って、すべての namespace 内のすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを設定します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. すべての namespace のすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを作成します。YAML を **web-allow-all-namespaces.yaml** ファイルに保存します。

```
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) デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

(2) すべての namespace のすべての Pod を選択します。

注記

デフォルトでは、**namespaceSelector** の指定を省略した場合、namespace は選択されません。つまり、ポリシーは、ネットワークポリシーがデプロイされている namespace からのトラフィックのみを許可します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**alpine** イメージを **secondary** namespace にデプロイし、シェルを開始します。

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

3. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

```
<!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. namespace からアプリケーションへのトラフィックを許可するネットワークポリシーの作成

特定の namespace からラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーを設定するには、次の手順に従います。以下の場合にこれを行うことができます。

- 運用データベースへのトラフィックを、運用ワークロードがデプロイされている namespace のみに制限します。
- 特定の namespace にデプロイされた監視ツールを有効にして、現在の namespace からメトリクスをスクレイピングします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. ラベルが **purpose=production** の特定の namespace 内にあるすべての Pod からのトラフィックを許可するポリシーを作成します。YAML を **web-allow-prod.yaml** ファイルに保存します。

```
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 デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

2 ラベルが **purpose=production** の namespace 内にある Pod のみにトラフィックを制限します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**prod** namespace を作成します。

```
$ oc create namespace prod
```

3. 次のコマンドを実行して、**prod** namespace にラベルを付けます。

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

4. 次のコマンドを実行して、**dev** namespace を作成します。

```
$ oc create namespace dev
```

5. 次のコマンドを実行して、**dev** namespace にラベルを付けます。

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

6. 次のコマンドを実行して、**alpine** イメージを **dev** namespace にデプロイし、シェルを開始します。

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

7. シェルで次のコマンドを実行し、リクエストがブロックされていることを確認します。

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

予想される出力

```
wget: download timed out
```

8. 次のコマンドを実行して、**alpine** イメージを **prod** namespace にデプロイし、シェルを開始します。

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

9. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

```
<!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. ネットワークポリシーの編集

namespace の既存のネットワークポリシーを編集できます。通常の編集には、ポリシーが適用される Pod、許可される Ingress トラフィック、トラフィックを受け入れる宛先ポートへの変更が含まれる場合があります。**apiVersion**、**kind**、および **name** フィールドは、リソース自体を定義するためのもので、**NetworkPolicy** オブジェクトを編集するときは、これらを変更しないでください。

4.3.1. ネットワークポリシーの編集

namespace のネットワークポリシーを編集できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

- ネットワークポリシーが存在する namespace で作業している。

手順

1. オプション: namespace のネットワークポリシーオブジェクトをリスト表示するには、以下のコマンドを入力します。

```
$ oc get networkpolicy
```

ここでは、以下のようになります。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

2. ネットワークポリシーオブジェクトを編集します。

- ネットワークポリシーの定義をファイルに保存した場合は、ファイルを編集して必要な変更を加えてから、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

<policy_file>
ネットワークポリシーを含むファイルの名前を指定します。

- ネットワークポリシーオブジェクトを直接更新する必要がある場合、以下のコマンドを入力できます。

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

3. ネットワークポリシーオブジェクトが更新されていることを確認します。

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

4.3.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
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 オブジェクトの名前。
2. ポリシーが適用される Pod を説明するセレクター。
3. ポリシーオブジェクトが入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namaspace にある Pod を照合して検索します。
4. トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

4.4. ネットワークポリシーの削除

namespace からネットワークポリシーを削除できます。

4.4.1. CLI を使用したネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- ネットワークポリシーオブジェクトを削除するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

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

4.5. ネットワークポリシーの表示

namespace のネットワークポリシーを表示するには、次の手順を使用します。

4.5.1. CLI を使用したネットワークポリシーの表示

namespace のネットワークポリシーを検査できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- namespace のネットワークポリシーを一覧表示します。
  - namespace で定義されたネットワークポリシーオブジェクトを表示するには、以下のコマンドを実行します。

    ```
    $ oc get networkpolicy
    ```

  - オプション: 特定のネットワークポリシーを検査するには、以下のコマンドを入力します。

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

    ここでは、以下のようになります。

    <policy_name>
    検査するネットワークポリシーの名前を指定します。

    <namespace>
    オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

    以下に例を示します。

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

oc describe コマンドの出力

```
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 複数ネットワーク

5.1. 複数ネットワークの使用について

デフォルトの OVN-Kubernetes Container Network Interface (CNI) プラグインに加えて、MicroShift Multus CNI を使用して他の CNI プラグインをチェーンすることもできます。MicroShift Multus のインストールおよび使用はオプションです。

5.1.1. MicroShift の追加ネットワーク

クラスターのインストール中、設定をカスタマイズしない限り、**default** の Pod ネットワークはデフォルト値で設定されます。デフォルトのネットワークは、クラスターの通常のネットワークトラフィックをすべて処理します。MicroShift Multus CNI プラグインを使用すると、他のネットワークの Pod に追加のインターフェイスを追加できます。これにより、スイッチングやルーティングなどのネットワーク機能を提供する Pod を設定する際に柔軟性が得られます。

5.1.1.1. ネットワーク分離のための追加ネットワークをサポート

MicroShift 4.17 では、次の追加ネットワークがサポートされています。

- Bridge: 同じホスト上の Pod が相互に、またホストと通信できるようにします。
- IPVLAN: ホスト上の Pod が他のホストと通信できるようにします。
  - これは、MACVLAN ベースの追加ネットワークに似ています。
  - MACVLAN ベースの追加ネットワークとは異なり、各 Pod は親物理ネットワークインターフェイスと同じ MAC アドレスを共有します。
- MACVLAN: ホスト上の Pod が物理ネットワークインターフェイスを使用して他のホストや他のホスト上の Pod と通信できるようにします。MACVLAN ベースの追加ネットワークに接続された各 Pod には、一意の MAC アドレスが提供されます。

注記

追加のネットワークに対するネットワークポリシーの設定はサポートされていません。

5.1.1.2. 使用例: ネットワーク分離のための追加ネットワーク

コントロールプレーンとデータプレーンの分離など、ネットワークの分離が必要な状況では、追加のネットワークを使用できます。たとえば、Pod がホスト上のネットワークにアクセスし、エッジにデプロイされたデバイスと通信できるようにする場合は、追加のインターフェイスを設定できます。これらのエッジデバイスは、分離された Operator ネットワーク上にあるか、定期的に切断される可能性があります。

トラフィックの分離は、以下のようなパフォーマンスおよびセキュリティー関連の理由で必要になります。

パフォーマンス

2 つの異なるプレーンでトラフィックを送信して、各プレーンのトラフィック量を管理できます。

セキュリティー

機密トラフィックは、セキュリティー上の考慮に基づいて管理されているネットワークに送信でき、テナントまたはカスタマー間で共有できないプライベートを分離できます。

重要

Multus CNI プラグインは、MicroShift サービスの起動時にデプロイされます。したがって、MicroShift の起動後に **microshift-multus** RPM パッケージを追加する場合は、ホストの再起動が必要です。再起動すると、すべてのコンテナーが Multus アノテーション付きで再作成されます。

5.1.1.3. 追加ネットワークの実装方法

クラスターのすべての Pod はクラスター全体のデフォルトネットワークを依然として使用し、クラスター全体での接続性を維持します。すべての Pod には、クラスター全体の Pod ネットワークに割り当てられる **eth0** インターフェイスがあります。

- **oc get pod <pod_name> -o=jsonpath='{ .metadata.annotations.k8s\.v1\.cni\.cncf\.io/network-status }'** コマンドを使用して、Pod のインターフェイスを表示できます。
- MicroShift Multus CNI を使用するネットワークインターフェイスを追加する場合、その名前は **net1**、**net2**、...、**netN** です。
- CNI 設定は、MicroShift Multus DaemonSet の起動時に作成されます。この設定は自動生成され、デフォルトの委譲であるプライマリー CNI が含まれます。MicroShift の場合、デフォルトの CNI は OVN-Kubernetes です。

5.1.1.4. Pod に追加のネットワークを接続する方法

Pod に追加のネットワークインターフェイスを接続するには、インターフェイスの接続方法を定義する設定を作成して適用する必要があります。

- 使用する追加のネットワークを設定する必要があります。ネットワークにはそれぞれ違いがあるため、デフォルトの設定はありません。
- **NetworkAttachmentDefinition** カスタムリソース (CR) を使用して各インターフェイスを指定するには、YAML マニフェストを適用する必要があります。これらの各 CR 内の設定は、そのインターフェイスがどのように作成されるかを定義します。
- CRI-O は Multus を使用するように設定する必要があります。デフォルト設定は **microshift-multus** RPM に含まれています。
    - Multus CNI が既存の MicroShift インスタンスにインストールされている場合は、ホストを再起動する必要があります。
    - Multus CNI が MicroShift と一緒にインストールされている場合は、CR と Pod を追加してから MicroShift サービスを開始できます。このシナリオではホストを再起動する必要はありません。

5.1.1.5. 追加のネットワークタイプの設定

次のセクションでは、追加のネットワークの具体的な設定フィールドを説明します。

5.1.2. 実行中のクラスターへの Multus CNI プラグインのインストール

高性能ネットワーク設定のために Pod に追加のネットワークを接続する場合は、MicroShift Multus RPM パッケージをインストールできます。インストール後、Multus アノテーションを持つすべての Pod を再作成するには、ホストを再起動する必要があります。

重要

Multus CNI プラグインのアンインストールはサポートされていません。

前提条件

1. ホストへの root アクセス権限がある。

手順

1. 次のコマンドを実行して、Multus RPM パッケージをインストールします。

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

ヒント

ここで追加のネットワーク用のカスタムリソース (CR) を作成すると、1 回の再起動でインストールを完了し、設定を適用できます。

2. パッケージマニフェストをアクティブなクラスターに適用するには、次のコマンドを実行してホストを再起動します。

```
$ sudo systemctl restart
```

検証

1. 再起動後、次のコマンドを実行して、Multus CNI プラグインコンポーネントが作成されていることを確認します。

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

出力例

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

次のステップ

1. まだ行っていない場合は、使用する追加のネットワークを設定して適用します。
2. 作成された CR を使用するアプリケーションをデプロイします。

5.1.3. ブリッジネットワークの追加設定

次のオブジェクトは、Bridge CNI プラグインの設定パラメーターを説明します。

表5.1 Bridge CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	**string**	CNI 仕様のバージョン。値 **0.4.0** が必要です。

フィールド	型	説明
type	**string**	設定する CNI プラグインの名前: **bridge**。
ipam	**object**	IPAM CNI プラグインの設定オブジェクト。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。
bridge	**string**	オプション: 使用する仮想ブリッジの名前を指定します。ブリッジインターフェイスがホストに存在しない場合は、これが作成されます。デフォルト値は **cni0** です。
ipMasq	**boolean**	オプション: 仮想ネットワークから外すトラフィックの IP マスカレードを有効にするには、**true** に設定します。すべてのトラフィックの送信元 IP アドレスは、ブリッジの IP アドレスに書き換えられます。ブリッジに IP アドレスがない場合は、この設定は影響を与えません。デフォルト値は **false** です。
isGateway	**boolean**	オプション: IP アドレスをブリッジに割り当てるには **true** に設定します。デフォルト値は **false** です。
isDefaultGateway	**boolean**	オプション: ブリッジを仮想ネットワークのデフォルトゲートウェイとして設定するには、**true** に設定します。デフォルト値は **false** です。**isDefaultGateway** が **true** に設定される場合、**isGateway** も自動的に **true** に設定されます。
forceAddress	**boolean**	オプション: 仮想ブリッジの事前に割り当てられた IP アドレスの割り当てを許可するには、**true** に設定します。**false** に設定される場合、重複サブセットの IPv4 アドレスまたは IPv6 アドレスが仮想ブリッジに割り当てられるとエラーが発生します。デフォルト値は **false** です。
hairpinMode	**boolean**	オプション: 仮想ブリッジが受信時に使用した仮想ポートでイーサネットフレームを送信できるようにするには、**true** に設定します。このモードは、**Reflective Relay** (リフレクティブリレー) としても知られています。デフォルト値は **false** です。
promiscMode	**boolean**	オプション: ブリッジで無作為検出モード (Promiscuous Mode) を有効にするには、**true** に設定します。デフォルト値は **false** です。
mtu	**integer**	オプション: 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
enabledad	**boolean**	オプション: コンテナー側の **veth** の重複アドレス検出を有効にします。デフォルト値は **false** です。
macspoofchk	**boolean**	オプション: MAC スプーフィングチェックを有効にして、コンテナーから発信されるトラフィックをインターフェイスの MAC アドレスに制限します。デフォルト値は **false** です。

5.1.3.1. ブリッジ CNI プラグインの設定例

次の例では、MicroShift Multus CNI で使用するために **bridge-conf** という名前の追加ネットワークを設定します。

```
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. ipvlan 追加ネットワークの設定

以下のオブジェクトは、IPVLAN CNI プラグインの設定パラメーターを説明しています。

表5.2 IPVLAN CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	**string**	CNI 仕様のバージョン。値 **0.3.1** が必要です。
name	**string**	CNO 設定に以前に指定した **name** パラメーターの値。
type	**string**	設定する CNI プラグインの名前: **ipvlan**。
ipam	**object**	IPAM CNI プラグインの設定オブジェクト。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。これは、プラグインが連鎖している場合を除き必要です。
mode	**string**	オプション: 仮想ネットワークの操作モードを指定します。この値は、**l2**、**l3**、または **l3s** である必要があります。デフォルト値は **l2** です。

フィールド	型	説明
master	**string**	オプション: ネットワーク割り当てに関連付けるイーサネットインターフェイスを指定します。**master** が指定されない場合、デフォルトのネットワークルートのインターフェイスが使用されます。
mtu	**integer**	オプション: 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
linkInContainer	**boolean**	オプション: マスターインターフェイスが、コンテナーネットワーク namespace とメインネットワーク namespace のどちらにあるかを指定します。コンテナー namespace マスターインターフェイスの使用を要求するには、値を **true** に設定します。

重要

- **ipvlan** オブジェクトは、仮想インターフェイスが **master** インターフェイスと通信することを許可しません。したがって、コンテナーは **ipvlan** インターフェイスを使用してホストに到達できません。コンテナーが、Precision Time Protocol (**PTP**) をサポートするネットワークなど、ホストへの接続を提供するネットワークに参加していることを確認してください。
- 1 つのの **master** インターフェイスを、**macvlan** と **ipvlan** の両方を使用するように同時に設定することはできません。
- インターフェイスに依存できない IP 割り当てスキームの場合、**ipvlan** プラグインは、このロジックを処理する以前のプラグインと連鎖させることができます。**master** が省略された場合、前の結果にはスレーブにする **ipvlan** プラグインのインターフェイス名が 1 つ含まれていなければなりません。**ipam** が省略された場合、**ipvlan** インターフェイスの設定には前の結果が使用されます。

5.1.4.1. IPVLAN CNI プラグインの設定例

以下の例では、**ipvlan-net** という名前の追加のネットワークを設定します。

```
{
  "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"
      }
    ]
  }
}
```

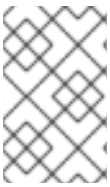
-

5.1.5. macvlan 追加ネットワークの設定

以下のオブジェクトは、MACVLAN CNI プラグインの設定パラメーターを説明しています。

表5.3 MACVLAN CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	**string**	CNI 仕様のバージョン。値 **0.3.1** が必要です。
name	**string**	CNO 設定に以前に指定した **name** パラメーターの値。
type	**string**	設定する CNI プラグインの名前: **macvlan**。
ipam	**object**	IPAM CNI プラグインの設定オブジェクト。プラグインは、アタッチメント定義への IP アドレスの割り当てを管理します。
mode	**string**	オプション: 仮想ネットワークのトラフィックの可視性を設定します。**bridge**、**passthru**、**private**、または **vepa** のいずれかである必要があります。値が指定されない場合、デフォルト値は **bridge** になります。
master	**string**	オプション: 新しく作成された macvlan インターフェイスに関連付けるホストネットワークインターフェイス。値が指定されていない場合は、デフォルトのルートインターフェイスが使用されます。
mtu	**string**	オプション: 指定された値への最大転送単位 (MTU)。デフォルト値はカーネルによって自動的に設定されます。
linkInContainer	**boolean**	オプション: マスターインターフェイスが、コンテナーネットワーク namespace とメインネットワーク namespace のどちらにあるかを指定します。コンテナー namespace マスターインターフェイスの使用を要求するには、値を **true** に設定します。



注記

プラグイン設定の **master** キーを指定する場合は、競合の可能性を回避するために、プライマリーネットワークプラグインに関連付けられているものとは異なる物理ネットワークインターフェイスを使用してください。

5.1.5.1. MACVLAN CNI プラグイン設定の例

以下の例では、**macvlan-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
```

```
  "linkInContainer": false,
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
    }
}
```

5.1.6. 関連情報

- 複数のネットワークの設定と使用

5.2. 複数のネットワークの設定と使用

MicroShift Multus Container Network Interface (CNI) をインストールした後、設定を使用して他のネットワークプラグインを使用できます。

5.2.1. IP アドレス管理タイプと追加ネットワーク

IP アドレスは、設定した IP アドレス管理 (IPAM) CNI プラグインを通じて追加のネットワークにプロビジョニングされます。MicroShift でサポートされている IP アドレスのプロビジョニングタイプは **host-local**、**static**、および **dhcp** です。

5.2.1.1. ブリッジインターフェイスの詳細

bridge タイプのインターフェイスと **dhcp** IPAM を使用する場合は、ブリッジネットワークでリッスンする DHCP サーバーが必要です。ファイアウォールを使用している場合は、ネットワークゾーンで DHCP トラフィックを許可するために、**firewall-cmd --remove-service=dhcp** コマンドを実行して、**firewalld** サービスを設定することも必要です。

5.2.1.2. macvlan インターフェイスの詳細

macvlan タイプのインターフェイスは、ホストが接続されているネットワークにアクセスします。つまり、**dhcp** IPAM プラグインが使用されている場合、インターフェイスがホストネットワーク上の DHCP サーバーから IP アドレスを受信できます。

5.2.1.3. ipvlan インターフェイスの詳細

ipvlan インターフェイスは、ホストネットワークにも直接アクセスできますが、ホストインターフェイスと MAC アドレスを共有します。共有 MAC アドレスにより、**ipvlan** タイプのインターフェイスは、**dhcp** プラグインでは使用できません。IPAM プラグインは、**ClientID** を使用した DHCP プロトコルをサポートしていません。

5.2.2. 追加ネットワーク用の NetworkAttachmentDefinition の作成

追加のネットワークの **NetworkAttachmentDefinition** 設定ファイルを作成するには、次の手順に従います。この例では、bridge-type インターフェイスを使用します。また、**host-local** IP アドレス管理 (IPAM) を使用するここでのサンプルワークフローを使用して、サポートされているその他の追加ネットワークタイプを設定することもできます。

重要

bridge と **dhcp** IPAM を使用する場合は、ブリッジされたネットワークでリッスンする DHCP サーバーが必要です。ファイアウォールも使用している場合は、ネットワークゾーンで DHCP トラフィックを許可するように firewalld サービスを設定することも必要です。この場合は、**firewall-cmd --remove-service=dhcp** コマンドを実行できます。

前提条件

- MicroShift Multus CNI がインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- クラスターが実行されている。

手順

1. オプション: 次のコマンドを実行して、MicroShift クラスターが Multus CNI で実行されていることを確認します。

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

出力例

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

2. 次のコマンドを実行し、次のサンプルファイルを参照して **NetworkAttachmentDefinition** 設定ファイルを作成します。

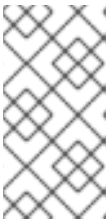
```
$ oc apply -f network-attachment-definition.yaml
```

NetworkAttachmentDefinition のファイルの例

```
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** 値は CNI プラグインの名前を指定します。この例では **bridge** タイプを使用します。
2. **bridge** 値は、使用されている MicroShift ホスト上のブリッジの名前です。Pod の追加インターフェイスはそのブリッジに接続されます。ホスト上にインターフェイスが存在しない場合は、Bridge CNI によって作成されます。インターフェイスがすでに存在する場合は再利用されます。以下の例では、インターフェイスの名前は **br-test** です。
3. IPAM タイプ。
4. IPv6 アドレスをセカンダリーインターフェイスに追加できます。



注記

ブリッジの名前の使用は、プラグインの **bridge** タイプに固有です。他のプラグインは **NetworkAttachmentDefinitions** で異なるフィールドを使用します。たとえば、**macvlan** および **ipvlan** 設定は **master** を使用して、割り当てるホストインターフェイスを指定します。

5.2.3. Pod の追加ネットワークへの追加

Pod を追加のネットワークに追加できます。Pod が作成されると、追加のネットワークが Pod に接続されます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

すでに実行中の Pod に追加のネットワークを接続する場合は、Pod を再起動する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターが実行されている。
- Pod を接続する **NetworkAttachmentDefinition** オブジェクトによって定義されたネットワークが存在します。

手順

1. **Pod** YAML ファイルにアノテーションを追加します。以下のアノテーション形式のいずれかのみを使用できます。
   a. カスタマイズせずに追加ネットワークを割り当てるには、以下の形式でアノテーションを追加します。**<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。

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

1 **<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。複数の追加ネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じ追加ネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークに割り当てます。

ブリッジタイプの追加ネットワークのアノテーションの例

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

b. カスタマイズして追加のネットワークを割り当てるには、以下の形式でアノテーションを追加します。

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

1 **NetworkAttachmentDefinition** オブジェクトによって定義される追加のネットワークの名前を指定します。

2 **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。

3 オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. **Pod** YAML ファイルを作成し、追加のネットワークの **NetworkAttachmentDefinition** アノテーションを追加するには、次のコマンドを実行し、サンプル YAML を使用します。

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

1 **<test-bridge>** を、使用する Pod 名に置き換えます。

出力例

```
pod/test_bridge created
```

test-bridge Pod YAML の例

```
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. **NetworkAttachmentDefinition** アノテーションが正しいことを確認します。

NetworkAttachmentDefinition アノテーションの例

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

4. オプション: **NetworkAttachmentDefinition** アノテーションが **Pod** YAML に存在することを確認するには、**<name>** を Pod の名前に置き換えて次のコマンドを実行します。

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

(1) **<name>** を、使用する Pod 名に置き換えます。以下の例では、test-bridge が使用されます。

次の例では、**test-bridge** が **net1** の追加ネットワークに接続されています。

```
$ oc get pod <test_bridge> -o yaml
```

出力例

```
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** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod に割り当てられる追加のネットワークのステータスを説明します。アノテーションの値はプレーンテキストの値として保存されます。

5. 以下のコマンドを使用して、Pod が実行されていることを確認します。

```
$ oc get pod
```

出力例

```
NAME        READY   STATUS    RESTARTS   AGE
test_bridge   1/1     Running   0          81s
```

5.2.4. 追加のネットワークの設定

NetworkAttachmentDefinition オブジェクトを作成して適用した後、次の例の手順に従って追加のネットワークを設定します。この例では、**bridge** タイプの追加ネットワークが使用されます。このワークフローは、他の追加のネットワークタイプにも使用できます。

前提条件

1. **NetworkAttachmentDefinition** オブジェクト設定を作成して適用しました。

手順

1. 以下のコマンドを実行して、ブリッジがホストに作成されていることを確認します。

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

出力例

```
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. 次のコマンドを実行して、ブリッジの IP アドレスを設定します。

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

3. 次のコマンドを実行して、IP アドレス設定がブリッジに追加されたことを確認します。

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

出力例

```
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 アドレスは想定どおりに設定されています。

4. 次のコマンドを実行して、Pod の IP アドレスを確認します。

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

出力例

```
[{
    "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) ブリッジの追加ネットワークは想定どおりに接続されます。

5. オプション: **oc exec** を使用して Pod にアクセスし、**ip** コマンドを使用してそのインターフェイスを確認できます。

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

出力例

```
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) 予想どおり、Pod は **net1 インターフェイス** 上の 10.10.0.20 IP アドレスに接続されます。

6. MicroShift ホストから Pod 内の HTTP サーバーにアクセスして、接続が期待どおりに機能していることを確認します。以下のコマンドを使用します。

```
$ curl 10.10.0.20:8080
```

出力例

```
Hello MicroShift
```

5.2.5. 追加ネットワークからの Pod の削除

Pod を削除するだけで、追加のネットワークから Pod を削除できます。

前提条件

- 追加のネットワークが Pod に割り当てられている。
- OpenShift CLI (**oc**) がインストールされている。
- クラスターにログインする。

手順

- Pod を削除するには、以下のコマンドを入力します。

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

  - **<name>** は Pod の名前です。
  - **<namespace>** は Pod が含まれる namespace です。

5.2.6. Multus ネットワークのトラブルシューティング

複数のネットワークの設定が適切に設定されていない場合、Pod が起動に失敗する可能性があります。次の手順は、いくつかの一般的なシナリオを解決するのに役立ちます。

5.2.6.1. Pod のネットワークを設定できない

Multus CNI プラグインが Pod にネットワークアノテーションを適用できない場合、Pod は起動しません。追加のネットワーク CNI のいずれかが失敗した場合、Pod も起動に失敗する可能性があります。

エラーの例

```
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
```

この場合、CNI 障害を解決するには次の手順を実行できます。

- **NetworkAttachmentDefinitions** とアノテーションの両方の値を確認します。
- アノテーションを削除して、デフォルトネットワークのみで Pod が正常に作成されたかどうかを確認します。そうでない場合は、Multus 設定以外のネットワークの問題を示している可能性があります。
- デバイス管理者の場合は、**kubelet** によって生成されたログに特に注意しながら、**crio.service** または **microshift.service** ログを確認してください。

たとえば、**kubelet** からの以下のエラーは、プライマリー CNI が実行されていないことを示しています。この状況は、Pod が起動していないこと、または **cni_default_network** 設定が正しくないことなどの CRI-O の設定ミスが原因で発生する可能性があります。

kubelet が生成したエラーの例

```
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. 設定ファイルがない

アノテーションが存在しない **NetworkAttachmentDefinition** 設定 YAML を参照しているため、Pod を作成できない場合があります。この場合、通常次のようなエラーが発生します。

ログの例

```
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"`
```

エラー出力の例

```
"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"
```

このエラーを修正するには、**NetworkAttachmentDefinitions** YAML を作成し、適用します。

5.2.7. 関連情報

- 複数ネットワークの使用について
- 追加ネットワークの IP アドレス割り当ての設定

第6章 ルートの設定

クラスター用に MicroShift のルートを設定できます。

6.1. HTTP ベースのルートの作成

ルートを使用すると、公開された URL でアプリケーションをホストできます。これは、アプリケーションのネットワークセキュリティー設定に応じて、セキュリティー保護または保護なしを指定できます。HTTP ベースのルートとは、セキュアではないルートで、基本的な HTTP ルーティングプロトコルを使用してセキュリティー保護されていないアプリケーションポートでサービスを公開します。

以下の手順では、**hello-openshift** アプリケーションを例に、Web アプリケーションへのシンプルな HTTP ベースのルートを作成する方法を説明します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- MicroShift クラスターにアクセスできる。
- あるポートを公開する Web アプリケーションと、そのポートでトラフィックをリッスンする TCP エンドポイントがあります。

手順

1. 以下のコマンドを実行して、**hello-openshift** というサービスを作成します。

```
$ oc expose pod hello-microshift -n $namespace
```

2. 次のコマンドを実行して、**hello-openshift** アプリケーションに対して、セキュアではないルートを作成します。

```
$ oc expose svc/hello-microshift --hostname=microshift.com $namespace
```

検証

- 以下のコマンドを実行して、**route** リソースが作成されたことを確認します。

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

1. この例では、ルートの名前は **hello-microshift** で、namespace の名前は **hello-microshift** です。

上記で作成されたセキュアでないルートの YAML 定義

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-microshift
  namespace: hello-microshift
spec:
  host: microshift.com 1
```

```
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-microshift
```

1 ホスト名の例。

2 **targetPort** は、ルーターがサービスのエンドポイントポートをマッピングするために必要です。

注記

MicroShift は、デフォルトの Ingress ドメインを作成する API を使用せず、代わりに自動生成されたドメインにワイルドカードを提供します。各ルートは個別のホスト名を定義することもできます。

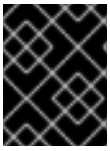
6.2. HTTP STRICT TRANSPORT SECURITY

HTTP Strict Transport Security (HSTS) ポリシーは、HTTPS トラフィックのみがルートホストで許可されるブラウザークライアントに通知するセキュリティーの拡張機能です。また、HSTS は、HTTP リダイレクトを使用せずに HTTPS トランスポートにシグナルを送ることで Web トラフィックを最適化します。HSTS は Web サイトとの対話を迅速化するのに便利です。

HSTS ポリシーが適用されると、HSTS はサイトから Strict Transport Security ヘッダーを HTTP および HTTPS 応答に追加します。HTTP を HTTPS にリダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用できます。HSTS を強制している場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するため、リダイレクトの必要がなくなります。

クラスター管理者は、以下を実行するために HSTS を設定できます。

- ルートごとに HSTS を有効にします。
- ルートごとに HSTS を無効にします。
- ドメインごとに HSTS を適用するか、ドメインと組み合わせた namespace ラベルを使用します。



重要

HSTS はセキュアなルート (edge-terminated または re-encrypt) でのみ機能します。この設定は、HTTP またはパススルールートには適していません。

6.3. ルートごとの HTTP STRICT TRANSPORT SECURITY の有効化

HTTP 厳密なトランスポートセキュリティー (HSTS) は HAProxy テンプレートに実装され、**haproxy.router.openshift.io/hsts_header** アノテーションを持つ edge および re-encrypt ルートに適用されます。

前提条件

- クラスターへの root アクセス権限がある。

- OpenShift CLI (**oc**) がインストールされている。

手順

- ルートで HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge-terminated または re-encrypt ルートに追加します。これを実行するには、**oc annotate** ツールを使用してこれを実行できます。

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

1 この例では、最長期間は **31536000** ミリ秒 (約 8.5 時間) に設定されます。

注記

この例では、等号 (**=**) が引用符で囲まれています。これは、annotate コマンドを正しく実行するために必要です。

アノテーションで設定されたルートの例

```
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 必須。**max-age** は、HSTS ポリシーが有効な期間 (秒単位) を測定します。**0** に設定すると、これはポリシーを無効にします。

2 任意。**includeSubDomains** は、クライアントに対し、ホストのすべてのサブドメインにホストと同じ HSTS ポリシーを持つ必要があることを指示します。

3 任意。**max-age** が 0 より大きい場合、**preload** を **haproxy.router.openshift.io/hsts_header** に追加し、外部サービスがこのサイトをそれぞれの HSTS プリロードリストに含めることができます。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザーはこれらのリストを使用し、サイトと対話する前でも HTTPS 経由で通信できるサイトを判別できます。**preload** を設定していない場合、ブラウザーはヘッダーを取得するために、HTTPS を介してサイトと少なくとも 1 回対話している必要があります。

6.3.1. ルートごとの HTTP Strict Transport Security の無効化

ルートごとに HSTS (HTTP Strict Transport Security) を無効にするには、ルートアノテーションの **max-age** の値を **0** に設定します。

前提条件

- クラスターへの root アクセス権限がある。
- OpenShift CLI (**oc**) がインストールされている。

手順

- HSTS を無効にするには、以下のコマンドを入力してルートアノテーションの **max-age** の値を **0** に設定します。

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

ヒント

または、以下の YAML を適用して config map を作成できます。

ルートごとに HSTS を無効にする例

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

- namespace のすべてのルートで HSTS を無効にするには、following コマンドを入力します。

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

検証

1. すべてのルートのアノテーションをクエリーするには、以下のコマンドを入力します。

```
$ 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}}'
```

出力例

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

6.3.2. ドメインごとに HTTP Strict Transport Security の強制

準拠した HSTS ポリシーアノテーションを使用してルートを設定できます。準拠しない HSTS ルートを持つアップグレードされたクラスターを処理するには、ソースでマニフェストを更新し、更新を適用できます。

oc expose route コマンドまたは **oc create route** コマンドを使用して、HSTS を強制するドメインにルートを追加することはできません。このコマンドの API はアノテーションを受け入れないためです。

重要

HSTS は安全でないルート、つまり TLS 以外のルートには適用できません。

前提条件

- クラスターへの root アクセス権限がある。
- OpenShift CLI (**oc**) がインストールされている。

手順

- 次の **oc annotate コマンド** を実行して、クラスター内のすべてのルートに HSTS を適用します。

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

- 次の **oc annotate コマンド** を実行して、特定の namespace 内のすべてのルートに HSTS を適用します。

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

1 _<my_namespace> は、使用する namespace に置き換えます。

検証

- 以下のコマンドを実行して、すべてのルートで HSTS アノテーションを確認します。

```
$ 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}}'
```

出力例

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

6.4. スループットの問題のトラブルシューティング方法

Red Hat build of MicroShift を使用してデプロイされたアプリケーションによって、特定のサービス間の待機時間が異常に長くなるなど、ネットワークスループットの問題が発生する場合があります。

Pod のログが問題の原因を指摘しない場合は、以下の方法を使用してパフォーマンスの問題を分析します。

- **ping** または **tcpdump** などのパケットアナライザーを使用して Pod とそのノード間のトラフィックを分析します。
  たとえば、問題を生じさせる動作を再現している間に各ノードで **tcpdump** ツールを実行 します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。ノードインターフェイスが他の Pod、ストレージデバイス、またはデータプレーンからのトラフィックで過負荷になると、Red Hat build of MicroShift で遅延が発生する可能性があります。

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

1 **podip** は Pod の IP アドレスです。 **oc get pod <pod_name> -o wide** コマンドを実行して Pod の IP アドレスを取得します。

  tcpdump は、これらの 2 つの Pod 間のすべてのトラフィックが含まれる **/tmp/dump.pcap** のファイルを生成します。ファイルサイズを最小限に抑えるために問題を再現するすぐ前と問題を再現したすぐ後ににアナライザーを実行することが良いでしょう。次のコマンドを使用して、ノード間でパケットアナライザーを実行 することもできます。

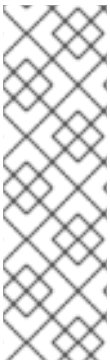
```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- ストリーミングのスループットおよび UDP スループットを測定するために **iperf** などの帯域幅測定ツールを使用します。最初に Pod からツールを実行し、次にノードから実行して、ボトルネックを特定します。

6.5. COOKIE の使用によるルートのステートフル性の維持

Red Hat build of MicroShift は、すべてのトラフィックを同じエンドポイントにヒットさせることによりステートフルなアプリケーションのトラフィックを可能にするスティッキーセッションを提供します。ただし、エンドポイント Pod が再起動、スケーリング、または設定の変更などによって終了する場合、このステートフル性はなくなります。

Red Hat build of MicroShift は Cookie を使用してセッションの永続化を設定できます。Ingress Controller はユーザー要求を処理するエンドポイントを選択し、そのセッションの Cookie を作成します。Cookie は要求の応答として戻され、ユーザーは Cookie をセッションの次の要求と共に送り返します。Cookie は Ingress Controller に対し、セッションを処理しているエンドポイントを示し、クライアント要求が Cookie を使用して同じ Pod にルーティングされるようにします。



注記

Cookie は、HTTP トラフィックを表示できないので、パススルールートで設定できません。代わりに、送信元 IP アドレスをベースに数が計算され、バックエンドを判断します。

バックエンドが変わると、トラフィックが間違ったサーバーに転送されてしまい、スティッキーではなくなります。送信元 IP を非表示にするロードバランサーを使用している場合は、すべての接続に同じ番号が設定され、トラフィックは同じ Pod に送られます。

6.5.1. Cookie を使用したルートのアノテーション

ルート用に自動生成されるデフォルト名を上書きするために Cookie 名を設定できます。これにより、ルートトラフィックを受信するアプリケーションが Cookie 名を認識できるようになります。Cookie を削除すると、次の要求でエンドポイントの再選択が強制的に実行される可能性があります。その結果、

サーバーがオーバーロードしている場合は、クライアントからの要求を取り除き、それらの再分配を試行します。

手順

1. 指定される cookie 名でルートにアノテーションを付けます。

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

ここでは、以下のようになります。

<route_name>
Pod の名前を指定します。

<cookie_name>
cookie の名前を指定します。

たとえば、ルート **my_route** に cookie 名 **my_cookie** でアノテーションを付けるには、以下を実行します。

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

2. 変数でルートのホスト名を取得します。

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

ここでは、以下のようになります。

<route_name>
Pod の名前を指定します。

3. cookie を保存してからルートにアクセスします。

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

ルートに接続する際に、直前のコマンドによって保存される cookie を使用します。

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

6.6. パスベースのルート

パスベースのルートは、URL に対して比較できるパスコンポーネントを指定します。この場合、ルートのトラフィックは HTTP ベースである必要があります。そのため、それぞれが異なるパスを持つ同じホスト名を使用して複数のルートを提供できます。ルーターは、最も具体的なパスの順に基づいてルートと一致する必要があります。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティーを示しています。

表6.1 ルートの可用性

ルート	比較対象	アクセス可能
www.example.com/test	www.example.com/test	はい
	www.example.com	いいえ
www.example.com/test および www.example.com	www.example.com/test	はい
	www.example.com	はい
www.example.com	www.example.com/text	Yes (ルートではなく、ホストで一致)
	www.example.com	はい

パスが 1 つでセキュリティー保護されていないルート

```
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 パスは、パスベースのルートに唯一追加される属性です。

注記

ルーターは TLS を終了させず、要求のコンテンツを読み込みことができないので、パスベースのルーティングは、パススルー TLS を使用する場合には利用できません。

6.7. HTTP ヘッダーの設定

ヘッダーを設定または削除するときに、個別のルートを使用してリクエストヘッダーとレスポンスヘッダーを変更できます。ルートアノテーションを使用して特定のヘッダーを設定することもできます。ヘッダーを設定するさまざまな方法は、連携時に課題となる可能性があります。

注記

ヘッダーを設定または削除できるのは、**Route** CR 内のみです。ヘッダーは、追加できません。HTTP ヘッダーに値が設定されている場合、その値は完全である必要があるため、今後追加する必要はありません。X-Forwarded-For ヘッダーなどのヘッダーを追加することが適切な状況では、**spec.httpHeaders.actions** の代わりに **spec.httpHeaders.forwardedHeaderPolicy** フィールドを使用します。

Route 仕様の例

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

ルートオーバーライド値で定義されたアクションはすべて、ルートアノテーションを使用して設定されます。

6.7.1. 特殊なケースのヘッダー

次のヘッダーは、設定または削除が完全に禁止されているか、特定の状況下で許可されています。

ヘッダー名	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
proxy	いいえ	**プロキシー** HTTP リクエストヘッダーを使用して、ヘッダー値を **HTTP_PROXY** 環境変数に挿入して、脆弱な CGI アプリケーションを悪用できます。**プロキシー** HTTP リクエストヘッダーも標準ではないため、設定中にエラーが発生しやすくなります。	いいえ
host	はい	**IngressController** CR を使用して **ホスト** HTTP 要求ヘッダーが設定されている場合、HAProxy は正しいルートを検索するときに失敗する可能性があります。	いいえ
strict-transport-security	いいえ	**strict-transport-security** HTTP 応答ヘッダーはルートアノテーションを使用してすでに処理されているため、別の実装は必要ありません。	はい: **haproxy.router.openshift.io/hsts_header** ルートアノテーション

ヘッダー名	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
cookie と **set-cookie**	いいえ	HAProxy が設定する Cookie は、クライアント接続を特定のバックエンドサーバーにマップするセッション追跡に使用されます。これらのヘッダーの設定を許可すると、HAProxy のセッションアフィニティーが妨げられ、HAProxy の Cookie の所有権が制限される可能性があります。	はい: * **haproxy.router.openshift.io/disable_cookie** ルートアノテーション * **haproxy.router.openshift.io/cookie_name** ルートアノテーション

6.8. ルート内の HTTP リクエストおよびレスポンスヘッダーの設定または削除

コンプライアンス目的またはその他の理由で、特定の HTTP 要求および応答ヘッダーを設定または削除できます。これらのヘッダーは、Ingress Controller によって提供されるすべてのルート、または特定のルートに対して設定または削除できます。

たとえば、ルートを提供する Ingress Controller によってデフォルトのグローバルな場所が指定されている場合でも、コンテンツが複数の言語で記述されていると、Web アプリケーションが特定のルートの別の場所でコンテンツを提供するように指定できます。

以下の手順では Content-Location HTTP リクエストヘッダーを設定するルートを作成し、アプリケーション (**https://app.example.com**) に URL が関連付けられ、**https://app.example.com/lang/en-us** のロケーションにダイレクトされるようにします。アプリケーショントラフィックをこの場所にダイレクトすると、特定のルートを使用する場合はすべて、アメリカ英語で記載された Web コンテンツにアクセスすることになります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- Red Hat build of MicroShift クラスターにプロジェクト管理者としてログインしている。
- あるポートを公開する Web アプリケーションと、そのポートでトラフィックをリッスンする HTTP または TCP エンドポイントがある。

手順

1. ルート定義を作成し、**app-example-route.yaml** というファイルに保存します。

   HTTP ヘッダーディレクティブを使用して作成されたルートの YAML 定義

   ```
   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. HTTP ヘッダーに対して実行するアクションのリスト。
2. 変更するヘッダーのタイプ。この場合は、応答ヘッダーです。
3. 変更するヘッダーの名前。設定または削除できる使用可能なヘッダーのリストは、**HTTP ヘッダーの設定** を参照してください。
4. ヘッダーに対して実行されるアクションのタイプ。このフィールドには、**Set** または **Delete** の値を指定できます。
5. HTTP ヘッダーの設定時は、**値** を指定する必要があります。値は、そのヘッダーで使用可能なディレクティブのリストからの文字列 (例: **DENY)** にすることも、HAProxy の動的値構文を使用して解釈される動的値にすることもできます。この場合、値はコンテンツの相対位置に設定されます。

2. 新しく作成したルート定義を使用して、既存の Web アプリケーションへのルートを作成します。

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

HTTP リクエストヘッダーの場合、ルート定義で指定されたアクションは、Ingress Controller の HTTP リクエストヘッダーに対して実行されたアクションの後に実行されます。これは、ルート内のこれらのリクエストヘッダーに設定された値が、Ingress Controller に設定された値よりも優先されることを意味します。HTTP ヘッダーの処理順序の詳細は、**HTTP ヘッダーの設定** を参照してください。

6.9. INGRESS オブジェクトを使用したルートの作成

一部のエコシステムコンポーネントには Ingress リソースとの統合機能がありますが、ルートリソースとは統合しません。このケースに対応するために、Red Hat build of MicroShift は、Ingress オブジェクトが作成されるときに、管理対象ルートオブジェクトを自動的に作成します。これらのルートオブジェクトは、対応する Ingress オブジェクトが削除されると削除されます。

手順

1. Red Hat build of MicroShift コンソールまたは **oc create** コマンドを実行して Ingress オブジェクトを定義します。

Ingress の YAML 定義

```
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** アノテーションは、**Route** の **spec.tls.termination** フィールドを設定するために使用できます。**Ingress** にはこのフィールドがありません。許可される値は **edge**、**passthrough**、および **reencrypt** です。その他のすべての値は警告なしに無視されます。アノテーション値が設定されていない場合は、**edge** がデフォルトルートになります。デフォルトのエッジルートを実装するには、TLS 証明書の詳細をテンプレートファイルで定義する必要があります。

(3) **Ingress** オブジェクトを操作する場合、ルートを操作する場合とは異なり、明示的なホスト名を指定する必要があります。**<host_name>.<cluster_ingress_domain>** 構文 (**apps.openshiftdemos.com** など) を使用して、***.<cluster_ingress_domain>** ワイルドカード DNS レコードとクラスターのサービング証明書を利用できます。それ以外の場合は、選択したホスト名の DNS レコードがあることを確認する必要があります。

a. **route.openshift.io/termination** アノテーションで **passthrough** の値を指定する場合は、仕様で **path** を **''** に設定し、**pathType** を **ImplementationSpecific** に設定します。

```
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** を Ingress オブジェクトで使用して、カスタム宛先証明書 (CA) でルートを定義できます。アノテーションは、生成されたルートに挿入される kubernetes シークレット **secret-ca-cert** を参照します。

   a. Ingress オブジェクトから宛先 CA を使用してルートオブジェクトを指定するには、シークレットの **data.tls.crt** 指定子に PEM エンコード形式の証明書を使用して **kubernetes.io/tls** または **Opaque** タイプのシークレットを作成する必要があります。

2. ルートを一覧表示します。

```
$ oc get routes
```

結果には、**frontend-** で始まる名前の自動生成ルートが含まれます。

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

このルートを検査すると、以下のようになります。

自動生成されるルートの YAML 定義

```
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
```

6.10. INGRESS オブジェクトを介してデフォルトの証明書を使用してルートを作成する

TLS 設定を指定せずに Ingress オブジェクトを作成すると、Red Hat build of MicroShift によって安全でないルートが生成されます。デフォルトの Ingress 証明書を使用してセキュアなエッジ終端ルートを生成する Ingress オブジェクトを作成するには、次のように空の TLS 設定を指定できます。

前提条件

- 公開したいサービスがあります。
- OpenShift CLI (**oc**) にアクセスできる。

手順

1. Ingress オブジェクトの YAML ファイルを作成します。この例では、ファイルの名前は **example-ingress.yaml** です。

   Ingress オブジェクトの YAML 定義

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:
  rules:
    ...
  tls:
  - {} (1)
```

   (1) この正確な構文を使用して、カスタム証明書を指定せずに TLS を指定します。

2. 次のコマンドを実行して、Ingress オブジェクトを作成します。

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

検証

- 次のコマンドを実行して、Red Hat build of MicroShift が Ingress オブジェクトの想定されるルートを作成したことを確認します。

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

出力例

```
apiVersion: v1
```

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

(1) ルートの名前には、Ingress オブジェクトの名前とそれに続くランダムな接尾辞が含まれます。

(2) デフォルトの証明書を使用するには、ルートで **spec.certificate** を指定しないでください。

(3) ルートは、**edge** の終了ポリシーを指定する必要があります。

6.11. INGRESS アノテーションでの宛先 CA 証明書を使用したルート作成

route.openshift.io/destination-ca-certificate-secret アノテーションを Ingress オブジェクトで使用して、カスタム宛先 CA 証明書でルートを定義できます。

前提条件

- PEM エンコードされたファイルで証明書/キーのペアを持つことができます。ここで、証明書はルートホストに対して有効となっています。
- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- PEM エンコードされたファイルの別の宛先 CA 証明書が必要です。
- 公開する必要のあるサービスが必要です。

手順

1. **route.openshift.io/destination-ca-certificate-secret** を Ingress アノテーションに追加します。

```
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) アノテーションは kubernetes シークレットを参照します。

2. このアノテーションで参照されているシークレットは、生成されたルートに挿入されます。

出力例

```
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-----
...
```

6.12. セキュリティー保護されたルート

セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。OpenShift Container Platform ドキュメントへの次のリンクでは、カスタム証明書を使用して再暗号化、エッジ、およびパススルールートを作成する方法を説明しています。

- カスタム証明書を使用した re-encrypt ルートの作成
- カスタム証明書を使用した edge ルートの作成
- passthrough ルートの作成

第7章 ファイアウォールの使用

MicroShift ではファイアウォールは必要ありませんが、ファイアウォールを使用すると、MicroShift API への望ましくないアクセスを防ぐことができます。

7.1. ファイアウォールを通過するネットワークトラフィックについて

Firewalld は、バックグラウンドで実行され、接続要求に応答して、動的にカスタマイズ可能なホストベースのファイアウォールを作成するネットワークサービスです。Red Hat Enterprise Linux for Edge (RHEL for Edge) を MicroShift とともに使用している場合は、通常、firewalld がすでにインストールされているため、firewalld を設定するだけで済みます。詳細は、次の手順で説明します。全体として、**firewalld** サービスの実行中に次の OVN-Kubernetes トラフィックを明示的に許可する必要があります。

CNI Pod から CNI Pod へ

CNI Pod からホストネットワーク Pod/ホストネットワーク Pod からホストネットワーク Pod

CNI Pod

CNI ネットワークを使用する Kubernetes Pod

ホストネットワーク Pod

ホストネットワークを使用する Kubernetes Pod 次の手順を使用して、**firewalld** サービスを設定できます。ほとんどの場合、firewalld は RHEL for Edge インストールに含まれています。firewalld がインストールされなていない場合は、このセクションにある簡単な手順でインストールできます。

重要

MicroShift Pod は、内部 CoreDNS コンポーネントおよび API サーバーにアクセスできる必要があります。

関連情報

- 必要なファイアウォール設定
- ファイアウォールを介したネットワークトラフィックの許可

7.2. FIREWALLD サービスのインストール

RHEL for Edge を使用している場合は、firewalld をインストールする必要があります。サービスを使用するには、設定を行うだけです。firewalld がない場合で、firewalld を使用したい場合は、次の手順を使用できます。

次の手順を使用して、MicroShift の **firewalld** サービスをインストールして実行します。

手順

1. オプション: 以下のコマンドを実行して、システムで firewalld があるかを確認します。

```
$ rpm -q firewalld
```

2. **firewalld** サービスがインストールされていない場合は、次のコマンドを実行します。

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

3. ファイアウォールを開始するには、次のコマンドを実行します。

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

7.3. 必要なファイアウォール設定

クラスターネットワークの IP アドレス範囲は、ファイアウォールの設定時に有効にする必要があります。デフォルト値を使用するか、IP アドレス範囲をカスタマイズできます。デフォルトの **10.42.0.0/16** 設定からクラスターネットワークの IP アドレス範囲をカスタマイズする場合は、ファイアウォール設定でも同じカスタム範囲を使用する必要があります。

表7.1 ファイアウォールの IP アドレス設定

IP 範囲	ファイアウォールルールが必要	説明
10.42.0.0/16	いいえ	他の Pod へのホストネットワーク Pod アクセス
169.254.169.1	はい	Red Hat build of MicroShift API サーバーへのホストネットワーク Pod アクセス

ファイアウォール構成に必須の設定コマンドの例を次に示します。

コマンドの例

- 他の Pod へのホストネットワーク Pod アクセスを設定します。

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

- Red Hat build of MicroShift API などのホストエンドポイントによってバックアップされたサービスへのホストネットワーク Pod アクセスを設定します。

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

7.4. オプションのポート設定の使用

MicroShift ファイアウォールサービスでは、オプションのポート設定が可能です。

手順

- カスタマイズされたポートをファイアウォール設定に追加するには、次のコマンド構文を使用します。

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

表7.2 オプションのポート

ポート	プロトコル	説明
80	TCP	OpenShift Container Platform ルーターを介してアプリケーションを提供するために使用される HTTP ポート。
443	TCP	OpenShift Container Platform ルーターを介してアプリケーションを提供するために使用される HTTPS ポート。
5353	UDP	OpenShift Container Platform ルート mDNS ホストに応答する mDNS サービス。
30000-32767	TCP	NodePort サービス用に予約されたポート範囲。LAN 上のアプリケーションを公開するために使用できます。
30000-32767	UDP	NodePort サービス用に予約されたポート範囲。LAN 上のアプリケーションを公開するために使用できます。
6443	TCP	Red Hat build of MicroShift API の HTTPS API ポート。

以下は、API サーバーのポート 6443 など、MicroShift で実行されているサービスへのファイアウォールを介した外部アクセスを必要とする場合に使用されるコマンドの例です。たとえば、ルーターを介して公開されるアプリケーションのポート 80 および 443 です。

コマンドの例

- MicroShift API サーバーのポートを設定します。

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

MicroShift インスタンスの不要なポートを閉じるには、「ネットワークセキュリティーを強化するために不使用または不要なポートの閉鎖」の手順に従ってください。

関連情報

- ネットワークのセキュリティーを強化するための不使用または不要なポートの閉鎖

7.5. ポートを開くためのサービスの追加

MicroShift インスタンスでは、**firewall-cmd** コマンドを使用してポートでサービスを開くことができます。

手順

1. オプション: 次のコマンドを実行すると、firewalld 内のすべての事前定義サービスを表示できます。

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

2. デフォルトのポートで必要なサービスを開くには、次のコマンド例を実行します。

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

7.6. ファイアウォールを介したネットワークトラフィックの許可

IP アドレス範囲を設定し、Pod からネットワークゲートウェイを通過する内部トラフィックを許可する DNS サーバーを挿入することで、ファイアウォールを通過するネットワークトラフィックを許可できます。

手順

1. 次のコマンドのいずれかを使用して、IP アドレス範囲を設定します。
   a. 次のコマンドを実行して、IP アドレス範囲をデフォルト値で設定します。

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

   b. 次のコマンドを実行して、カスタム値を使用して IP アドレス範囲を設定します。

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

2. Pod からの内部トラフィックがネットワークゲートウェイを通過できるようにするには、次のコマンドを実行します。

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

7.6.1. ファイアウォール設定の適用

ファイアウォール設定を適用するには、手順1だけからなる以下の手順を使用します。

手順

- ファイアウォールを介したネットワークアクセスの設定が完了したら、次のコマンドを実行してファイアウォールを再起動し、設定を適用します。

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

7.7. ファイアウォール設定の確認

ファイアウォールを再起動したら、設定を一覧表示して確認できます。

手順

- ポート関連のルールなど、デフォルトのパブリックゾーンに追加されたルールを確認するには、次のコマンドを実行します。

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

- 信頼されたゾーンに追加されたルール (IP 範囲関連のルールなど) を確認するには、次のコマンドを実行します。

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

7.8. サービスが公開されている場合のファイアウォールポートの概要

MicroShift でサービスを実行すると、firewalld がアクティブになることがよくあります。これにより、ポートへのトラフィックがファイアウォールによってブロックされる可能性があるため、MicroShift 上の特定のサービスが中断される可能性があります。ホストの外部から特定のサービスにアクセスできるようにする場合は、必要なファイアウォールポートが開いていることを確認する必要があります。ポートを開くには、いくつかの方法があります。

- **NodePort** および **LoadBalancer** タイプのサービスは、OVN-Kubernetes によって自動的に利用可能になります。
  このような場合、OVN-Kubernetes が iptables ルールを追加して、ノード IP アドレスへのトラフィックが関連するポートに配信されるようにします。これは PREROUTING ルールチェーンを使用して実行されます。トラフィックはローカルホストポートとサービスの firewalld ルールをバイパスするために OVN-K に転送されます。iptables および firewalld は、Red Hat Enterprise Linux (RHEL) 9 の nftables でサポートされています。iptables が生成する nftables ルールは、firewalld が生成するルールよりも常に優先されます。

- **HostPort** パラメーター設定を持つ Pod は自動的に使用可能になります。これには、ポート 80 と 443 を使用する **router-default** Pod も含まれます。
  HostPort Pod の場合、CRI-O 設定が iptables DNAT (宛先ネットワークアドレス変換) を Pod の IP アドレスとポートに設定します。

これらの方法は、クライアントが同じホスト上にあるかリモートホスト上にあるかに関係なく機能します。OVN-Kubernetes および CRI-O によって追加される iptables ルールは、PREROUTING チェーンと OUTPUT チェーンに割り当てられます。ローカルトラフィックは、インターフェイスが **lo** タイプに設定された OUTPUT チェーンを通過します。DNAT は、INPUT チェーンのフィラールールに到達する前に実行されます。

MicroShift API サーバーは CRI-O では実行されないため、ファイアウォール設定の影響を受けます。ファイアウォールでポート 6443 を開くと、MicroShift クラスター内の API サーバーにアクセスできます。

7.9. 関連情報

- RHEL: firewalld の使用と設定
- firewalld の現在の状況の表示

7.10. 既知のファイアウォールの問題

- ファイアウォールのリロードまたは再起動でトラフィックフローが中断されないようにするには、Red Hat Enterprise Linux (RHEL)を起動する前にファイアウォールコマンドを実行します。MicroShift の CNI ドライバーは、NodePort サービスを使用するトラフィックフローなど、一部のトラフィックフローに対して iptable ルールを使用します。iptable ルールは CNI ド

ライバーによって生成および挿入されますが、ファイアウォールのリロードまたは再起動時に削除されます。iptable ルールがないと、トラフィックフローが中断されます。MicroShift の実行後にファイアウォールコマンドを実行する必要がある場合は、**openshift-ovn-kubernetes** namespace で **ovnkube-master** Pod を手動で再起動して、CNI ドライバーによって制御されるルールをリセットします。

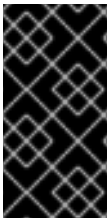
第8章 完全に切断されたホストのネットワーク設定

ネットワークのカスタマイズと設定を適用して、完全に切断されたホストで MicroShift を実行する方法を説明します。切断されたホストは、物理か仮想かに関係なく、ネットワーク接続なしで実行される Red Hat Enterprise Linux (RHEL) オペレーティングシステムバージョン 9.0 以降である必要があります。

8.1. 完全に切断されたホスト用のネットワークの準備

完全に切断されたオペレーティングシステムを実行しているデバイス上で MicroShift クラスターを起動して実行するには、次の手順を使用します。MicroShift ホストは、外部ネットワーク接続がない場合、完全に切断されているとみなされます。

通常、これは、サブネットを提供するためのネットワークインターフェイスコントローラー (NIC) がデバイスにアタッチされていないことを意味します。これらの手順は、セットアップ後に NIC が取り外されたホストでも実行できます。キックスタートファイルの **%post** フェーズを使用して、NIC を持たないホスト上でこれらの手順を自動化することもできます。



重要

MicroShift ではクラスター通信をサポートするためにネットワークデバイスが必要であるため、切断された環境のネットワークを設定することが必要です。この要件を満たすには、セットアップ中にシステムループバックデバイスに割り当てた "偽" の IP アドレスを使用するように MicroShift ネットワークを設定する必要があります。

8.1.1. 手順の概要

切断されたホストで MicroShift を実行するには、次の手順が必要です。

ホストの準備

- MicroShift が現在実行中の場合は停止し、サービスがネットワークに加えた変更をクリーンアップします。
- 永続的なホスト名を設定します。
- ループバックインターフェイスに ”偽” の IP アドレスを追加します。
- 偽の IP をローカルネームサーバーとして使用するように DNS を設定します。
- ホスト名のエントリーを **/etc/hosts** に追加します。

MicroShift 設定の更新

- **nodeIP** パラメーターを新しいループバック IP アドレスとして定義します。
- **.node.hostnameOverride** パラメーターを永続的なホスト名に設定します。

変更内容の有効化

- デフォルトの NIC がアタッチされている場合は無効にします。
- ホストまたはデバイスを再起動します。

MicroShift は起動後、クラスター内通信にループバックデバイスを使用して実行されます。

8.2. MICROSHIFT ネットワーク設定をデフォルトに戻す

MicroShift を停止してクリーンアップスクリプトを実行すると、ネットワークのカスタマイズを削除し、ネットワークをデフォルト設定に戻すことができます。

前提条件

- RHEL 9 以降。
- MicroShift 4.14 以降。
- ホスト CLI へのアクセスがある。

手順

1. 次のコマンドを実行して、MicroShift サービスを停止します。

```
$ sudo systemctl stop microshift
```

2. 次のコマンドを実行して、**kubepods.slice** systemd ユニットを停止します。

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

3. MicroShift は、OVN-K によって加えられたネットワークの変更を元に戻すためのヘルパースクリプトをインストールします。次のコマンドを入力して、クリーンアップスクリプトを実行します。

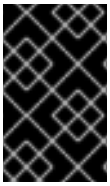
```
$ sudo /usr/bin/microshift-cleanup-data --ovn
```

8.3. 完全に切断されたホストのネットワーク設定

完全に切断されたホスト上で MicroShift を実行するためのネットワークを設定するには、ホストを準備し、ネットワーク設定を更新してから、再起動して新しい設定を適用する必要があります。すべてのコマンドはホスト CLI から実行されます。

前提条件

- RHEL 9 以降。
- MicroShift 4.14 以降。
- ホスト CLI へのアクセスがある。
- MicroShift の実行時に、内部 IP の競合と今後使用される外部 IP の潜在的な競合の両方を回避するために選択された有効な IP アドレス。
- MicroShift ネットワーク設定はデフォルトに設定されています。



重要

次の手順は、デバイスがフィールドにデプロイされた後に MicroShift クラスターへのアクセスが必要ないユースケースを対象としています。ネットワーク接続が削除された後は、リモートクラスターにアクセスできなくなります。

手順

1. 次のコマンドを実行して、偽の IP アドレスをループバックインターフェイスに追加します。

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

(1) この例で使用される偽の IP アドレスは “10.44.0.1” です。

注記

MicroShift の内部 IP の競合と今後使用される外部 IP の潜在的な競合の両方を回避できるのであれば、有効な IP はどれでも使用できます。たとえば、MicroShift ノードのサブネットと衝突しない、またはデバイス上の他のサービスによってアクセスされる任意のサブネットを使用することができます。

2. 自動 DNS を無視し、ローカルネームサーバーにリセットするように設定を変更して、ローカルネームサーバーを使用するように DNS インターフェイスを設定します。

   a. 次のコマンドを実行して自動 DNS をバイパスします。

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

   b. DNS インターフェイスがローカルネームサーバーを使用するように指示します。

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

3. 次のコマンドを実行して、デバイスのホスト名を取得します。

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

4. 次のコマンドを実行して、**/etc/hosts** ファイルにノードのホスト名のエントリーを追加します。

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

5. 次の YAML スニペットを **/etc/microshift/config.yaml** に追加して、MicroShift 設定ファイルを更新します。

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

6. これで、MicroShift はクラスター通信にループバックデバイスを使用できるようになりました。オフラインで使用するためのデバイスの準備を完了します。

   a. 現在デバイスに NIC がアタッチされている場合は、デバイスをネットワークから切断します。

   b. デバイスをシャットダウンし、NIC を切断します。

   c. オフライン設定を有効にするには、デバイスを再起動します。

7. 次のコマンドを実行して、MicroShift ホストを再起動し、設定の変更を適用します。

```
$ sudo systemctl reboot (1)
```

(1) この手順によりクラスターが再起動されます。検証を実装する前に、greenboot ヘルスチェックによってシステムが正常であると報告されるまで待ちます。

検証

この時点で、MicroShift ホストへのネットワークアクセスは切断されています。ホストターミナルにアクセスできる場合は、ホスト CLI を使用して、クラスターが安定した状態で開始されたことを確認できます。

1. 次のコマンドを入力して、MicroShift クラスターが実行されていることを確認します。

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

出力例

```
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
```