OpenShift Container Platform 4.3

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: 2020-11-13

OpenShift Container Platform 4.3 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

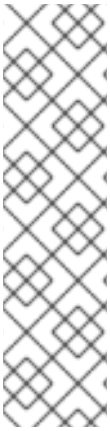
概要

この文書では、DNS、ingress および Pod ネットワークを含む、OpenShift Container Platform のクラスターネットワークを設定し、管理する方法を説明します。

目次

第1章 ネットワークについて

Kubernetes は、確実に Pod 間がネットワークで接続されるようにし、内部ネットワークから IP アドレスを各 Pod に割り当てます。これにより、Pod 内のすべてのコンテナーが同じホスト上に置かれているかのように動作します。各 Pod に IP アドレスを割り当てると、ポートの割り当て、ネットワーク、名前の指定、サービス検出、負荷分散、アプリケーション設定、移行などの点で、Pod を物理ホストや仮想マシンのように扱うことができます。



注記

一部のクラウドプラットフォームでは、169.254.169.254 IP アドレスでリッスンするメタデータ API があります。これは、IPv4 **169.254.0.0/16** CIDR ブロックのリンクローカル IP アドレスです。

この CIDR ブロックは Pod ネットワークから到達できません。これらの IP アドレスへのアクセスを必要とする Pod には、Pod 仕様の **spec.hostNetwork** フィールドを **true** に設定して、ホストのネットワークアクセスが付与される必要があります。

Pod ホストのネットワークアクセスを許可する場合、Pod に基礎となるネットワークインフラストラクチャーへの特権アクセスを付与します。

1.1. OPENSHIFT CONTAINER PLATFORM DNS

フロントエンドサービスやバックエンドサービスなど、複数のサービスを実行して複数の Pod で使用している場合、フロントエンド Pod がバックエンドサービスと通信できるように、ユーザー名、サービス IP などの環境変数を作成します。サービスが削除され、再作成される場合には、新規の IP アドレスがそのサービスに割り当てられるので、フロントエンド Pod がサービス IP の環境変数の更新された値を取得するには、これを再作成する必要があります。さらに、バックエンドサービスは、フロントエンド Pod を作成する前に作成し、サービス IP が正しく生成され、フロントエンド Pod に環境変数として提供できるようにする必要があります。

そのため、OpenShift Container Platform には DNS が組み込まれており、これにより、サービスは、サービス IP/ポートと共にサービス DNS によって到達可能になります。

第2章 ホストへのアクセス

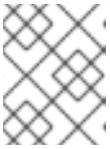
OpenShift Container Platform インスタンスにアクセスして、セキュアなシェル (SSH) アクセスでマスターノードにアクセスするために bastion ホストを作成する方法を学びます。

2.1. インストーラーでプロビジョニングされるインフラストラクチャークラスターでの AMAZON WEB SERVICES のホストへのアクセス

OpenShift Container Platform インストーラーは、OpenShift Container Platform クラスターにプロビジョニングされる Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのパブリック IP アドレスを作成しません。OpenShift Container Platform ホストに対して SSH を実行できるようにするには、以下の手順を実行する必要があります。

手順

1. **openshift-install** コマンドで作成される仮想プライベートクラウド (VPC) に対する SSH アクセスを可能にするセキュリティーグループを作成します。
2. インストーラーが作成したパブリックサブネットのいずれかに Amazon EC2 インスタンスを作成します。
3. パブリック IP アドレスを、作成した Amazon EC2 インスタンスに関連付けます。OpenShift Container Platform のインストールとは異なり、作成した Amazon EC2 インスタンスを SSH キーペアに関連付ける必要があります。これにはインターネットを OpenShift Container Platform クラスターの VPC にブリッジ接続するための SSH bastion としてのみの単純な機能しかないため、このインスタンスにどのオペレーティングシステムを選択しても問題ありません。どの Amazon Machine Image (AMI) を使用するかについては、注意が必要です。たとえば、Red Hat Enterprise Linux CoreOS では、インストーラーと同様に、Ignition でキーを指定することができます。
4. Amazon EC2 インスタンスをプロビジョニングし、これに対して SSH を実行した後に、OpenShift Container Platform インストールに関連付けた SSH キーを追加する必要があります。このキーは bastion インスタンスのキーとは異なる場合がありますが、異なるキーにしなければならない訳ではありません。



注記

直接の SSH アクセスは、障害復旧を目的とする場合にのみ推奨されます。Kubernetes API が応答する場合、特権付き Pod を代わりに実行します。

5. **oc get nodes** を実行し、出力を検査し、マスターであるノードのいずれかを選択します。ホスト名は **ip-10-0-1-163.ec2.internal** に類似したものになります。
6. Amazon EC2 に手動でデプロイした bastion SSH ホストから、そのマスターホストに対して SSH を実行します。インストール時に指定したものと同じ SSH キーを使用するようにします。

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

第3章 OPENSHIFT CONTAINER PLATFORM における CLUSTER NETWORK OPERATOR

Cluster Network Operator (CNO) は、インストール時にクラスター用に選択される Container Network Interface (CNI) デフォルトネットワークプロバイダープラグインを含む、OpenShift Container Platform クラスターの各種のクラスターネットワークコンポーネントをデプロイし、これらを管理します。

3.1. CLUSTER NETWORK OPERATOR

Cluster Network Operator は、**operator.openshift.io** API グループから **network** API を実装します。Operator は、DaemonSet を使用して OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグイン、またはクラスターのインストール時に選択したデフォルトネットワークプロバイダープラグインをデプロイします。

手順

Cluster Network Operator は、インストール時に Kubernetes **Deployment** としてデプロイされます。

1. 以下のコマンドを実行して Deployment のステータスを表示します。

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

出力例

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

2. 以下のコマンドを実行して、Cluster Network Operator の状態を表示します。

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

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.3.0     True        False         False      50m
```

以下のフィールドは、Operator のステータス (**AVAILABLE**、**PROGRESSING**、および **DEGRADED**) についての情報を提供します。**AVAILABLE** フィールドは、Cluster Network Operator が Available ステータス条件を報告する場合に **True** になります。

3.2. クラスターネットワーク設定の表示

すべての新規 OpenShift Container Platform インストールには、**cluster** という名前の **network.config** オブジェクトがあります。

手順

- **oc describe** コマンドを使用して、クラスターネットワーク設定を表示します。

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

出力例

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

1. **Spec** フィールドは、クラスターネットワークの設定済みの状態を表示します。
2. **Status** フィールドは、クラスターネットワークの現在の状態を表示します。

3.3. CLUSTER NETWORK OPERATOR のステータス表示

oc describe コマンドを使用して、Cluster Network Operator のステータスを検査し、その詳細を表示することができます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のステータスを表示します。

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

3.4. CLUSTER NETWORK OPERATOR ログの表示

oc logs コマンドを使用して、Cluster Network Operator ログを表示できます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のログを表示します。

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

重要

Open Virtual Networking (OVN) Kubernetes ネットワークプラグインは、テクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

OVN テクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/articles/4380121 を参照してください。

3.5. CLUSTER NETWORK OPERATOR (CNO) の設定

クラスターネットワークの設定は、Cluster Network Operator (CNO) 設定の一部として指定され、**cluster** という名前の CR オブジェクトに保存されます。CR は **operator.openshift.io** API グループの **Network** API のパラメーターを指定します。

defaultNetwork パラメーターのパラメーター値を CNO CR に設定することにより、OpenShift Container Platform クラスターのクラスターネットワーク設定を指定できます。以下の CR は、CNO のデフォルト設定を表示し、設定可能なパラメーターと有効なパラメーターの値の両方について説明しています。

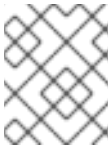
Cluster Network Operator CR

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork: 1
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: 2
  - 172.30.0.0/16
  defaultNetwork: 3
    ...
  kubeProxyConfig: 4
    iptablesSyncPeriod: 30s 5
    proxyArguments:
      iptables-min-sync-period: 6
      - 0s
```

1. Pod ID の割り当て、サブネットプレフィックスの長さの個別ノードへの割り当てに使用される IP アドレスのブロックを指定する一覧です。
2. サービスの IP アドレスのブロック。OpenShift SDN Container Network Interface (CNI) ネットワークプロバイダーは、サービスネットワークの単一 IP アドレスブロックのみをサポートします。
3. クラスターネットワークのデフォルトの CNI ネットワークプロバイダーを設定します。
4.

このオブジェクトのパラメーターは、Kubernetes ネットワークプロキシー (kube-proxy) 設定を指定します。OVN-Kubernetes デフォルト CNI ネットワークプロバイダーを使用している場合、

5 **iptables** ルールの更新期間。デフォルト値は **30s** です。有効なサフィックスには、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time ドキュメントで説明されています。



注記

OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、**iptablesSyncPeriod** パラメーターを調整する必要はなくなりました。

6 **iptables** ルールを更新する前の最小期間。このパラメーターにより、更新の頻度が高くなり過ぎないようにできます。有効なサフィックスには、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time で説明されています。

3.5.1. OpenShift SDN デフォルト CNI ネットワークプロバイダーの設定パラメーター

以下の YAML オブジェクトは、OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーの設定パラメーターについて説明しています。

注記

クラスターのインストール時にのみデフォルト CNI ネットワークプロバイダーの設定を変更することができます。

```
defaultNetwork:
  type: OpenShiftSDN 1
  openshiftSDNConfig: 2
    mode: NetworkPolicy 3
    mtu: 1450 4
    vxlanPort: 4789 5
```

1 使用されるデフォルト CNI ネットワークプロバイダープラグイン。

2 OpenShift SDN 固有の設定パラメーター。

3 OpenShiftSDN のネットワーク分離モード。

4 VXLAN オーバーレイネットワークの最大転送単位 (MTU)。通常、この値は自動的に設定されます。

5 すべての VXLAN パケットに使用するポート。デフォルト値は **4789** です。

3.5.2. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーター

以下の YAML オブジェクトは OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーターについて説明しています。

注記

クラスターのインストール時にのみデフォルト CNI ネットワークプロバイダーの設定を変更することができます。

```
defaultNetwork:
  type: OVNKubernetes (1)
  ovnKubernetesConfig: (2)
    mtu: 1400 (3)
    genevePort: 6081 (4)
```

(1) 使用されるデフォルト CNI ネットワークプロバイダープラグイン。

(2) OVN-Kubernetes 固有の設定パラメーター。

(3) Geneve (Generic Network Virtualization Encapsulation) オーバーレイネットワークの MTU。通常、この値は自動的に設定されます。

(4) Geneve オーバーレイネットワークの UDP ポート。

3.5.3. Cluster Network Operator の設定例

以下の例のように、CNO の完全な CR オブジェクトが表示されます。

Cluster Network Operator のサンプル CR

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  defaultNetwork:
    type: OpenShiftSDN
    openshiftSDNConfig:
      mode: NetworkPolicy
      mtu: 1450
      vxlanPort: 4789
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period:
      - 0s
```

第4章 OPENSHIFT CONTAINER PLATFORM の DNS OPERATOR

DNS Operator は、Pod に対して名前解決サービスを提供するために CoreDNS をデプロイし、これを管理し、OpenShift 内での DNS ベースの Kubernetes サービス検出を可能にします。

4.1. DNS OPERATOR

DNS Operator は、**operator.openshift.io** API グループから **dns** API を実装します。この Operator は、DaemonSet を使用して CoreDNS をデプロイし、DaemonSet の Service を作成し、 kubelet を Pod に対して名前解決に CoreDNS Service IP を使用するように指示するように設定します。

手順

DNS Operator は、インストール時に Kubernetes **Deployment** としてデプロイされます。

1. **oc get** コマンドを使用して Deployment ステータスを表示します。

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

出力例

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

ClusterOperator は、Operator の現在の状態を保持するカスタムリソースオブジェクトです。このオブジェクトは、Operator によってそれらの状態をクラスターの残りの部分に送るために使用されます。

2. **oc get** コマンドを使用して DNS Operator の状態を表示します。

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

出力例

```
NAME     VERSION     AVAILABLE   PROGRESSING   DEGRADED   SINCE
dns      4.1.0-0.11  True        False         False      92m
```

AVAILABLE、 **PROGRESSING** および **DEGRADED** は、Operator のステータスについての情報を提供します。**AVAILABLE** は、CoreDNS DaemonSet からの 1 つ以上の Pod が **Available** ステータス条件を報告する場合は **True** になります。

4.2. デフォルト DNS の表示

すべての新規 OpenShift Container Platform インストールには、**default** という名前の **dns.operator** があります。

手順

1. **oc describe** コマンドを使用してデフォルトの **dns** を表示します。

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

出力例

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

(1) 「Cluster Domain」フィールドは、完全修飾 Pod およびサービスドメイン名を作成するために使用されるベース DNS ドメインです。

(2) クラスター IP は、Pod が名前解決のためにクエリーするアドレスです。IP は、サービス CIDR 範囲の 10 番目のアドレスで定義されます。

2. クラスターのサービス CIDR を見つけるには、**oc get** コマンドを使用します。

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

出力例

```
[172.30.0.0/16]
```

4.3. DNS 転送の使用

DNS 転送を使用すると、指定のゾーンにどのネームサーバーを使用するかを指定することで、ゾーンごとに **etc/resolv.conf** で特定される転送設定をオーバーライドできます。

手順

1. **default** という名前の DNS Operator オブジェクトを変更します。

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

これにより、**Server** に基づく追加のサーバー設定ブロックを使用して **dns-default** という名前の ConfigMap を作成し、更新できます。クエリーに一致するゾーンを持つサーバーがない場合、名前解決は **/etc/resolv.conf** で指定されたネームサーバーにフォールバックします。

DNS の例

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: foo-server (1)
```

```
    zones: 2
      - foo.com
    forwardPlugin:
      upstreams: 3
        - 1.1.1.1
        - 2.2.2.2:5353
  - name: bar-server
    zones:
      - bar.com
      - example.com
    forwardPlugin:
      upstreams:
        - 3.3.3.3
        - 4.4.4.4:5454
```

1 **name** は、**rfc6335** サービス名の構文に準拠する必要があります。

2 **zones** は、**rfc1123** の **subdomain** の定義に準拠する必要があります。クラスタードメインの **cluster.local** は、 **zones** の無効な **subdomain** です。

3 **forwardPlugin** ごとに最大 15 の **upstreams** が許可されます。

注記

servers が定義されていないか、または無効な場合、ConfigMap にはデフォルトサーバーのみが含まれます。

2. ConfigMap を表示します。

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

以前のサンプル DNS に基づく DNS ConfigMap の例

```
apiVersion: v1
data:
  Corefile: |
    foo.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
        forward . 3.3.3.3 4.4.4.4:5454 1
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            policy sequential
```

```
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

1 **forwardPlugin** への変更により、CoreDNS DaemonSet のローリングアップデートがトリガーされます。

追加リソース

- DNS 転送の詳細は、CoreDNS forward のドキュメント を参照してください。

4.4. DNS OPERATOR のステータス

oc describe コマンドを使用して、DNS Operator のステータスを検査し、その詳細を表示することができます。

手順

DNS Operator のステータスを表示します。

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

4.5. DNS OPERATOR ログ

oc logs コマンドを使用して、DNS Operator ログを表示できます。

手順

DNS Operator のログを表示します。

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

第5章 OPENSHIFT CONTAINER PLATFORM の INGRESS OPERATOR

Ingress Operator は **ingresscontroller** API を実装し、OpenShift Container Platform クラスターサービスへの外部アクセスを可能にするコンポーネントです。Operator は、1 つ以上の HAProxy ベースの ingress コントローラー をデプロイし、管理してこれを可能にします。OpenShift Container Platform **Route** および Kubernetes **Ingress** リソースを指定して、トラフィックをルーティングするために Ingress Operator を使用します。

5.1. INGRESS 設定アセット

インストールプログラムでは、**config.openshift.io** API グループの **Ingress** リソースでアセットを生成します (**cluster-ingress-02-config.yml**)。

Ingress リソースの YAML 定義

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

インストールプログラムは、このアセットを **manifests/** ディレクトリーの **cluster-ingress-02-config.yml** ファイルに保存します。この **Ingress** リソースは、Ingress のクラスター全体の設定を定義します。この Ingress 設定は、以下のように使用されます。

- Ingress Operator は、クラスター Ingress 設定のドメインを、デフォルト Ingress コントローラーのドメインとして使用します。
- OpenShift API サーバー Operator は、クラスター Ingress 設定のドメインを、明示的なホストを指定しない **Route** リソースのデフォルトホストを生成する際に使用されるドメインとして使用します。

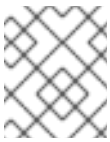
5.2. イメージコントローラー設定パラメーター

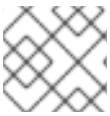
ingresscontrollers.operator.openshift.io リソースは以下の設定パラメーターを提供します。

パラメーター	説明

パラメーター	説明
domain	**domain** は Ingress コントローラーによって提供される DNS 名で、複数の機能を設定するために使用されます。 ● **LoadBalancerService** エンドポイント公開ストラテジーの場合、**domain** は DNS レコードを設定するために使用されます。**endpointPublishingStrategy** を参照してください。 ● 生成されるデフォルト証明書を使用する場合、証明書は **domain** およびその **subdomains** で有効です。**defaultCertificate** を参照してください。 ● この値は個別の Route ステータスに公開され、ユーザーは外部 DNS レコードのターゲット先を認識できるようにします。 **domain** 値はすべての Ingress コントローラーの中でも固有の値であり、更新できません。 空の場合、デフォルト値は **ingress.config.openshift.io/cluster .spec.domain** です。
replicas	**replicas** は Ingress コントローラーレプリカの必要な数です。設定されていない場合、デフォルト値は **2** になります。
endpointPublishingStrategy	**endpointPublishingStrategy** は Ingress コントローラーエンドポイントを他のネットワークに公開し、ロードバランサーの統合を有効にし、他のシステムへのアクセスを提供するために使用されます。 設定されていない場合、デフォルト値は **infrastructure.config.openshift.io/cluster .status.platform** をベースとします。 ● AWS: **LoadBalancerService** (外部スコープあり) ● Azure: **LoadBalancerService** (外部スコープあり) ● GCP: **LoadBalancerService** (外部スコープあり) ● その他: **HostNetwork**. **endpointPublishingStrategy** 値は更新できません。

パラメーター	説明
defaultCertificate	**defaultCertificate** 値は、Ingress コントローラーによって提供されるデフォルト証明書が含まれるシークレットの参照です。ルートが独自の証明書を指定しない場合、**defaultCertificate** が使用されます。 シークレットには以下のキーおよびデータが含まれる必要があります: * **tls.crt**: 証明書ファイルコンテンツ * **tls.key**: キーファイルコンテンツ 設定されていない場合、ワイルドカード証明書は自動的に生成され、使用されます。証明書は Ingress コントーラーの **domain** および **subdomains** で有効であり、生成された証明書 CA はクラスターの信頼ストアに自動的に統合されます。 使用中の証明書 (生成されるか、ユーザー指定の場合かを問わない) は OpenShift Container Platform のビルトイン OAuth サーバーに自動的に統合されます。
namespaceSelector	**namespaceSelector** は、Ingress コントローラーによってサービスされる namespace セットをフィルターするために使用されます。これはシャードの実装に役立ちます。
routeSelector	**routeSelector** は、Ingress コントローラーによって提供される Routes のセットをフィルターするために使用されます。これはシャードの実装に役立ちます。
nodePlacement	**nodePlacement** は、Ingress コントローラーのスケジュールに対する明示的な制御を有効にします。 設定されていない場合は、デフォルト値が使用されます。  **注記** **nodePlacement** パラメーターには、**nodeSelector** と **tolerations**の 2 つの部分が含まれます。以下は例になります。 nodePlacement: nodeSelector: matchLabels: beta.kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists

パラメーター	説明
tlsSecurityProfile	**tlsSecurityProfile** は、Ingress コントローラーの TLS 接続の設定を指定します。 これが設定されていない場合、デフォルト値は **apiservers.config.openshift.io/cluster** リソースをベースとして設定されます。 **Old**、**Intermediate**、および **Modern** のプロファイルタイプを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース **X.Y.Z** にデプロイされた **Intermediate** プロファイルを使用する仕様がある場合、リリース **X.Y.Z+1** へのアップグレードにより、新規のプロファイル設定が Ingress コントローラーに適用され、ロールアウトが生じる可能性があります。 Ingress コントローラーの最小の TLS バージョンは **1.1**で、最大の TLS バージョンは **1.2** です。  **重要** HAProxy Ingress コントローラーイメージは TLS **1.3** をサポートしません。**Modern** プロファイルには TLS**1.3** が必要であることから、これはサポートされません。Ingress Operator は **Modern** プロファイルを **Intermediate** に変換します。 また、Ingress Operator は TLS **1.0** の **Old** または **Custom** プロファイルを **1.1** に変換し、TLS**1.3** の **Custom** プロファイルを **1.2** に変換します。  **注記** 設定されたセキュリティープロファイルの暗号および最小の TLS バージョンが **TLSProfile** ステータスに反映されます。



注記

すべてのパラメーターはオプションです。

5.2.1. Ingress コントローラーの TLS プロファイル

tlsSecurityProfile パラメーターは、TLS セキュリティープロファイルのスキーマを定義します。このオブジェクトは、TLS セキュリティー設定をオペランドに適用するために Operator によって使用されます。

TLS セキュリティープロファイルには、以下の 4 つのタイプがあります。

- **Old**
- **Intermediate**
- **Modern**
- **Custom**

Old、**Intermediate**、および **Modern** プロファイルは 推奨される設定 をベースとします。**Custom** プロファイルは、個別の TLS セキュリティープロファイルパラメーターを指定する機能を提供します。

Old プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Old
```

Intermediate プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Intermediate
```

Modern プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Modern
```

Custom プロファイルは、ユーザーが定義する TLS セキュリティープロファイルです。

警告

無効な設定により問題が発生する可能性があるため、 **Custom** プロファイルを使用する際には注意してください。

Custom プロファイルのサンプル

```
spec:
  tlsSecurityProfile:
    type: Custom
    custom:
      ciphers:
        - ECDHE-ECDSA-AES128-GCM-SHA256
        - ECDHE-RSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```

5.2.2. Ingress コントローラーエンドポイントの公開ストラテジー

HostNetwork エンドポイント公開ストラテジーを持つ Ingress コントローラーには、ノードごとに単一の Pod レプリカのみを設定できます。**n** のレプリカを使用する場合、それらのレプリカをスケジュールできる **n** 以上のノードを使用する必要があります。各 Pod はスケジュールされるノードホストでポート **80** および **443** を要求するので、同じノードで別の Pod がそれらのポートを使用している場合、レプリカをノードにスケジュールすることはできません。

5.3. デフォルト INGRESS コントローラーの表示

Ingress Operator は、OpenShift Container Platform の中核となる機能であり、追加の設定なしに有効にできます。

すべての新規 OpenShift Container Platform インストールには、**ingresscontroller** の名前付きのデフォルトがあります。これは、追加の Ingress コントローラーで補足できます。デフォルトの **ingresscontroller** が削除される場合、Ingress Operator は 1 分以内にこれを自動的に再作成します。

手順

- デフォルト Ingress コントローラーを表示します。

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
```

5.4. INGRESS OPERATOR ステータスの表示

Ingress Operator のステータスを表示し、検査することができます。

手順

- Ingress Operator ステータスを表示します。

```
$ oc describe clusteroperators/ingress
```

5.5. INGRESS コントローラーログの表示

Ingress コントローラーログを表示できます。

手順

- Ingress コントローラーログを表示します。

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator
```

5.6. INGRESS コントローラーステータスの表示

特定の Ingress コントローラーのステータスを表示できます。

手順

- Ingress コントローラーのステータスを表示します。

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
```

5.7. カスタムデフォルト証明書の設定

管理者として、 Secret リソースを作成し、**IngressController** カスタムリソース (CR) を編集して Ingress コントローラーがカスタム証明書を使用するように設定できます。

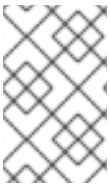
前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。ここで、証明書は信頼される認証局またはカスタム PKI で設定されたプライベートの信頼される認証局で署名されます。
- 証明書が Ingress ドメインに対して有効である必要があります。
- **IngressController** CR がなければなりません。デフォルトの CR を使用できます。

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

出力例

```
NAME      AGE
default   10m
```



注記

中間証明書がある場合、それらはカスタムデフォルト証明書が含まれるシークレットの **tls.crt** ファイルに組み込まれる必要があります。証明書を指定する際の順序は重要になります。サーバー証明書の後に中間証明書を一覧表示します。

手順

以下では、カスタム証明書とキーのペアが、現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提とします。**tls.crt** および **tls.key** を実際のパス名に置き換えます。さらに、 Secret リソースを作成し、これを IngressController CR で参照する際に、**custom-certs-default** を別の名前に置き換えます。

注記

このアクションにより、Ingress コントローラーはデプロイメントストラテジーを使用して再デプロイされます。

1. **tls.crt** および **tls.key** ファイルを使用して、カスタム証明書を含む Secret リソースを **openshift-ingress** namespace に作成します。

```
$ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt --
key=tls.key
```

2. IngressController CR を、新規証明書シークレットを参照するように更新します。

```
$ oc patch --type=merge --namespace openshift-ingress-operator ingresscontrollers/default \
  --patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
```

3. 更新が正常に行われていることを確認します。

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
  --output jsonpath='{.spec.defaultCertificate}'
```

出力例

```
map[name:custom-certs-default]
```

証明書シークレットの名前は、CR を更新するために使用された値に一致する必要があります。

IngressController CR が変更された後に、Ingress Operator はカスタム証明書を使用できるように Ingress コントローラーのデプロイメントを更新します。

5.8. INGRESS コントローラーのスケーリング

Ingress コントローラーは、スループットを増大させるための要件を含む、ルーティングのパフォーマンスや可用性に関する各種要件に対応するために手動でスケーリングできます。**oc** コマンドは、**IngressController** リソースをスケーリングするために使用されます。以下の手順では、デフォルトの **IngressController** をスケールアップする例を示します。

手順

1. デフォルト **IngressController**の現在の利用可能なレプリカ数を表示します。

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

出力例

```
2
```

2. **oc patch** コマンドを使用して、デフォルトの **IngressController** を必要なレプリカ数にスケーリングします。以下の例では、デフォルトの **IngressController** を 3 つのレプリカにスケーリングしています。

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":{"replicas":
3}}' --type=merge
```

出力例

```
ingresscontroller.operator.openshift.io/default patched
```

3. デフォルトの **IngressController** が指定したレプリカ数にスケーリングされていることを確認します。

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

出力例

```
3
```

注記

スケーリングは、必要な数のレプリカを作成するのに時間がかかるため、すぐに実行できるアクションではありません。

5.9. ルートラベルを使用した INGRESS コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセレクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

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

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

5.10. NAMESPACE ラベルを使用した INGRESS コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セレクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コン

トローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

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

出力例

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

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

5.11. 内部ロードバランサーを使用するように INGRESS コントローラーを設定する

クラウドプラットフォームで Ingress コントローラーを作成する場合、Ingress コントローラーはデフォルトでパブリッククラウドロードバランサーによって公開されます。管理者は、内部クラウドロードバランサーを使用する Ingress コントローラーを作成できます。

警告

クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも 1 つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。

重要

IngressController オブジェクトの**スコープ**を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

実装の詳細については、Kubernetes サービスドキュメント を参照してください。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

1. 以下の例のように、**<name>-ingress-controller.yaml** という名前のファイルに **IngressController** カスタムリソース (CR) を作成します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal 3
```

1. **<name>** を **IngressController** オブジェクトの名前に置き換えます。
2. コントローラーによって公開されるアプリケーションの**ドメイン**を指定します。
3. 内部ロードバランサーを使用するために **Internal** の値を指定します。

2. 以下のコマンドを実行して、直前の手順で定義された Ingress コントローラーを作成します。

```
$ oc create -f <name>-ingress-controller.yaml 1
```

1 **<name>** を **IngressController** オブジェクトの名前に置き換えます。

3. オプション: 以下のコマンドを実行して Ingress コントローラーが作成されていることを確認します。

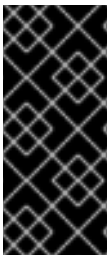
```
$ oc --all-namespaces=true get ingresscontrollers
```

5.12. クラスターを内部に配置するようにのデフォルト INGRESS コントローラーを設定する

削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定できます。

警告

クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも 1 つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。



重要

IngressController オブジェクトの**スコープ**を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

1. 削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定します。

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
```

```
    loadBalancer:
      scope: Internal
EOF
```

5.13. 追加リソース

- カスタム PKI の設定

第6章 ネットワークポリシー

6.1. ネットワークポリシーについて

クラスター管理者は、トラフィックをクラスター内の Pod に制限するネットワークポリシーを定義できます。

6.1.1. ネットワークポリシーについて

Kubernetes ネットワークポリシーをサポートする Kubernetes Container Network Interface (CNI) プラグインを使用するクラスターでは、ネットワークの分離は NetworkPolicy カスタムリソース (CR) オブジェクトによって完全に制御されます。OpenShift Container Platform 4.3 では、OpenShift SDN はデフォルトのネットワーク分離モードでの NetworkPolicy の使用をサポートしています。

注記

Kubernetes **v1** NetworkPolicy 機能は、Egress ポリシータイプおよび IPBlock 以外は OpenShift Container Platform で利用できます。

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。ホストのネットワークが有効にされている Pod は NetworkPolicy オブジェクトルールによる影響を受けません。

デフォルトで、プロジェクトのすべての 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。プロジェクトで 1 つ以上の Pod を分離するには、そのプロジェクトに NetworkPolicy オブジェクトを作成でき、許可される受信接続を指定できます。プロジェクト管理者は、各自のプロジェクト内で NetworkPolicy オブジェクトを作成し、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で一致する場合、Pod はそれらの 1 つ以上の NetworkPolicy オブジェクトで許可される接続のみを受け入れます。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。

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: []
```

- OpenShift Container Platform Ingress コントローラーからの接続のみを許可します。
プロジェクトで OpenShift Container Platform Ingress コントローラーからの接続のみを許可するには、以下の 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:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

Ingress コントローラーが **endpointPublishingStrategy: HostNetwork** で設定されている場合、Ingress コントローラー Pod はホストネットワーク上で実行されます。ホストネットワーク上で実行されている場合、Ingress コントローラーからのトラフィックに **netid:0** Virtual Network ID (VNID) が割り当てられます。Ingress Operator に関連付けられる namespace の **netid** は異なるため、**allow-from-openshift-ingress** ネットワークポリシーの **matchLabel** は **default** Ingress コントローラーからのトラフィックに一致しません。**default** namespace には **netid:0** VNID が割り当てられるため、**default** namespace に **network.openshift.io/policy-group: ingress** でラベルを付けて、**default** Ingress コントローラーからのトラフィックを許可できます。

- プロジェクト内の Pod からの接続のみを受け入れます。
Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下の 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たすことができます。

たとえば、先の例で定義された NetworkPolicy オブジェクトの場合、同じプロジェト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend**の付いた Pod は各ポリシーで許可されるすべての接続を受け入れます。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。

6.1.2. 次のステップ

- ネットワークポリシーの作成
- オプション: デフォルトネットワークポリシーの定義

6.1.3. 追加リソース

- マルチテナントネットワークポリシーの設定

6.2. ネットワークポリシーの作成

クラスター管理者は、namespace のネットワークポリシーを作成できます。

6.2.1. NetworkPolicy オブジェクトの作成

クラスターのプロジェクトに許可される Ingress 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、NetworkPolicy オブジェクトを作成できます。

前提条件

- クラスターは、NetworkPolicy オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用している（例 **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー）。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

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。**<policy-name>** はポリシールールを記述します。
   b. 作成したばかりのファイルで、以下の例のようなポリシーオブジェクトを定義します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: <policy-name> 1
spec:
  podSelector:
  ingress: []
```

   1 ポリシーオブジェクトの名前を指定します。

2. 以下のコマンドを実行してポリシーオブジェクトを作成します。

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

   以下の例では、新規 NetworkPolicy オブジェクトが **project1** という名前のプロジェクトに作成されます。

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

   出力例

```
networkpolicy "default-deny" created
```

6.2.2. 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: extensions/v1beta1
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 **name**。

2 ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。

3 ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。

4 トラフィックを受け入れる 1 つ以上の宛先の一覧。

6.3. ネットワークポリシーの表示

クラスター管理者は、namespace のネットワークポリシーを表示できます。

6.3.1. NetworkPolicy オブジェクトの表示

クラスターの NetworkPolicy オブジェクトを一覧表示できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

- クラスターで定義された NetworkPolicy 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
```

6.3.2. 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: extensions/v1beta1
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 **name**。

2 ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。

3 ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。

4 トラフィックを受け入れる 1 つ以上の宛先の一覧。

6.4. ネットワークポリシーの編集

クラスター管理者は、namespace の既存のネットワークポリシーを編集できます。

6.4.1. NetworkPolicy オブジェクトの編集

namespace の NetworkPolicy オブジェクトを編集できます。

前提条件

- クラスターは、NetworkPolicy オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用している（例 **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー）。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. オプション: 現在の NetworkPolicy オブジェクトを一覧表示します。
   a. 特定の namespace のポリシーオブジェクトを一覧表示するには、以下のコマンドを入力します。**<namespace>** をプロジェクトの namespace に置き換えます。

      ```
      $ oc get networkpolicy -n <namespace>
      ```

   b. クラスター全体についてのポリシーオブジェクトを一覧表示するには、以下のコマンドを入力します。

```
$ oc get networkpolicy --all-namespaces
```

2. NetworkPolicy オブジェクトを編集します。

   a. NetworkPolicy 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。**<policy-file>** を、オブジェクト定義が含まれるファイルの名前に置き換えます。

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

   b. NetworkPolicy 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。**<policy-name>** を NetworkPolicy オブジェクトの名前に置き換え、**<namespace>** をオブジェクトが存在するプロジェクトの名前に置き換えます。

   ```
   $ oc edit <policy-name> -n <namespace>
   ```

3. NetworkPolicy オブジェクトが更新されていることを確認します。**<namespace>** をオブジェクトが存在するプロジェクトの名前に置き換えます。

```
$ oc get networkpolicy -n <namespace> -o yaml
```

6.4.2. 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: extensions/v1beta1
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 **name**。

(2) ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。

(3) ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。

(4) トラフィックを受け入れる 1 つ以上の宛先の一覧。

6.4.3. 追加リソース

- ネットワークポリシーの作成

6.5. ネットワークポリシーの削除

クラスター管理者は、namespace からネットワークポリシーを削除できます。

6.5.1. NetworkPolicy オブジェクトの削除

NetworkPolicy オブジェクトを削除することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

- NetworkPolicy 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。**<policy-name>** をオブジェクトの名前に置き換えます。

```
$ oc delete networkpolicy <policy-name>
```

6.6. 新規プロジェクトのデフォルトネットワークポリシーの作成

クラスター管理者は、新規プロジェクトの作成時に NetworkPolicy オブジェクトを自動的に含めるように新規プロジェクトテンプレートを変更できます。新規プロジェクトのカスタマイズされたテンプレートがまだない場合には、まずテンプレートを作成する必要があります。

6.6.1. 新規プロジェクトのテンプレートの変更

クラスター管理者は、デフォルトのプロジェクトテンプレートを変更し、新規プロジェクトをカスタム要件に基づいて作成することができます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

手順

1. **cluster-admin** 権限を持つユーザーとしてのログイン。
2. デフォルトのプロジェクトテンプレートを生成します。

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

3. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで生成される **template.yaml** ファイルを変更します。
4. プロジェクトテンプレートは、**openshift-config** namespace に作成される必要があります。変更したテンプレートを読み込みます。

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

5. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。
   - Web コンソールの使用
     i. **Administration → Cluster Settings** ページに移動します。
     ii. **Global Configuration** をクリックし、すべての設定リソースを表示します。
     iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。
   - CLI の使用
     i. **project.config.openshift.io/cluster** リソースを編集します。

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

6. **spec** セクションを、**projectRequestTemplate** および **name** パラメーターを組み込むように更新し、アップロードされたプロジェクトテンプレートの名前を設定します。デフォルト名は **project-request** です。

   カスタムプロジェクトテンプレートを含むプロジェクト設定リソース

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

7. 変更を保存した後、変更が正常に適用されたことを確認するために、新しいプロジェクトを作成します。

6.6.2. 新規プロジェクトテンプレートへのネットワークポリシーオブジェクトの追加

クラスター管理者は、ネットワークポリシーオブジェクトを新規プロジェクトのデフォルトテンプレートに追加できます。OpenShift Container Platform は、プロジェクトのテンプレートに指定されたすべての NetworkPolicy CR を自動的に作成します。

前提条件

- クラスターは、NetworkPolicy オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用している（例 **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー）。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする必要があります。
- 新規プロジェクトのカスタムデフォルトプロジェクトテンプレートを作成している必要があります。

手順

1. 以下のコマンドを実行して、新規プロジェクトのデフォルトテンプレートを編集します。

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

<project_template> を、クラスターに設定したデフォルトテンプレートの名前に置き換えます。デフォルトのテンプレート名は **project-request** です。

2. テンプレートでは、各 NetworkPolicy オブジェクトを要素として **objects** パラメーターに追加します。**objects** パラメーターは、1つ以上のオブジェクトのコレクションを受け入れます。以下の例では、**objects** パラメーターのコレクションにいくつかの NetworkPolicy オブジェクトが含まれます。

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

3. オプション: 以下のコマンドを実行して、新規プロジェクトを作成し、ネットワークポリシーオブジェクトが正常に作成されることを確認します。

   a. 新規プロジェクトを作成します。

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

   1 **<project>** を、作成しているプロジェクトの名前に置き換えます。

   b. 新規プロジェクトテンプレートのネットワークポリシーオブジェクトが新規プロジェクトに存在することを確認します。

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

6.7. ネットワークポリシーを使用したマルチテナントモードの設定

クラスター管理者は、マルチテナントネットワークの分離を実行するようにネットワークポリシーを設定できます。

6.7.1. NetworkPolicy を使用したマルチテナント分離の設定

他のプロジェクト namespace の Pod およびサービスから分離できるようにプロジェクトを設定できます。

前提条件

- クラスターは、NetworkPolicy オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用している（例 **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. 以下の NetworkPolicy オブジェクトを作成します。
   a. **allow-from-openshift-ingress** という名前のポリシー:

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

   b. **allow-from-openshift-monitoring** という名前のポリシー。

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

```
  policyTypes:
  - Ingress
EOF
```

c. **allow-same-namespace** という名前のポリシー：

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

2. **default** Ingress コントローラー設定に **spec.endpointPublishingStrategy: HostNetwork** の値が設定されている場合、ラベルを **default** OpenShift Container Platform namespace に適用し、Ingress コントローラーとプロジェクト間のネットワークトラフィックを許可する必要があります。

a. **default** Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するかどうかを判別します。

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
  --output jsonpath='{.status.endpointPublishingStrategy.type}'
```

b. 直前のコマンドによりエンドポイント公開ストラテジーが **HostNetwork**として報告される場合には、**default** namespace にラベルを設定します。

```
$ oc label namespace default 'network.openshift.io/policy-group=ingress'
```

3. 以下のコマンドを実行し、NetworkPolicy オブジェクトが現在のプロジェクトに存在することを確認します。

```
$ oc get networkpolicy <policy-name> -o yaml
```

以下の例では、**allow-from-openshift-ingress** NetworkPolicy オブジェクトが表示されています。

```
$ oc get -n project1 networkpolicy allow-from-openshift-ingress -o yaml
```

出力例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
  namespace: project1
spec:
  ingress:
  - from:
```

```
  - namespaceSelector:
      matchLabels:
        network.openshift.io/policy-group: ingress
podSelector: {}
policyTypes:
- Ingress
```

6.7.2. 次のステップ

- デフォルトのネットワークポリシーの定義

第7章 複数ネットワーク

7.1. 複数ネットワークについて

Kubernetes では、コンテナーネットワークは Container Network Interface (CNI) を実装するネットワークプラグインに委任されます。

OpenShift Container Platform は、Multus CNI プラグインを使用して CNI プラグインのチェーンを許可します。クラスターのインストール時に、**デフォルト** の Pod ネットワークを設定します。デフォルトのネットワークは、クラスターのすべての通常のネットワークトラフィックを処理します。利用可能な CNI プラグインに基づいて **追加のネットワーク** を定義し、1 つまたは複数のネットワークを Pod に割り当てることができます。必要に応じて、クラスターの複数のネットワークを追加で定義することができます。これは、スイッチまたはルーティングなどのネットワーク機能を提供する Pod を設定する場合に柔軟性を実現します。

7.1.1. 追加ネットワークの使用シナリオ

データプレーンとコントロール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追加のネットワークを使用することができます。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

各プレーンのトラフィック量を管理するために、2 つの異なるプレーンにトラフィックを送信できます。

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することができます。

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** インターフェースがあります。Pod のインターフェースは、**oc exec -it <pod_name> -- ip a** コマンドを使用して表示できます。Multus CNI を使用するネットワークを追加する場合、それらの名前は **net1**、**net2**、...、 **netN** になります。

追加のネットワークを Pod に割り当てるには、インターフェースの割り当て方法を定義する設定を作成する必要があります。それぞれのインターフェースは、**NetworkAttachmentDefinition** タイプのあるカスタムリソース (CR) を使用して指定します。これらの CR のそれぞれにある CNI 設定は、インターフェースの作成方法を定義します。

7.1.2. OpenShift Container Platform の追加ネットワーク

OpenShift Container Platform は、クラスターに追加のネットワークを作成するために使用する以下の CNI プラグインを提供します。

- **bridge**: ブリッジベースの追加ネットワークを作成することで、同じホストにある Pod が相互に、かつホストと通信できます。
- **host-device**: ホストデバイスの追加ネットワークを作成することで、Pod がホストシステム上の物理イーサネットネットワークデバイスにアクセスすることができます。
- **macvlan**: macvlan ベースの追加ネットワークを作成することで、ホスト上の Pod が物理ネットワークインターフェースを使用して他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加ネットワークに割り当てられる各 Pod には固有の MAC アドレスが割り当てられます。

- **ipvlan**: ipvlan ベースの追加ネットワークを作成することで、macvlan ベースの追加ネットワークと同様に、ホスト上の Pod が他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加のネットワークとは異なり、各 Pod は親の物理ネットワークインターフェースと同じ MAC アドレスを共有します。

- **SR-IOV**: SR-IOV ベースの追加ネットワークを作成することで、Pod を ホストシステム上の SR-IOV 対応ハードウェアの仮想機能 (VF) インターフェースに割り当てることができます。

7.2. POD の追加のネットワークへの割り当て

クラスターユーザーとして、Pod を追加のネットワークに割り当てることができます。

7.2.1. 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 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。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当てることはできません。

前提条件

- Pod が追加ネットワークと同じ namespace にあること。
- OpenShift CLI (**oc**) のインストール。
- クラスターにログインすること。

手順

1. アノテーションを Pod オブジェクトに追加します。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。

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

   1 複数の追加ネットワークを指定するには、各ネットワークをカンマで区切ります。カ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ースをそのネットワークに割り当てます。

   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。

   ```
   metadata:
     annotations:
       k8s.v1.cni.cncf.io/networks: |-
         [
           {
             "name": "<network>", 1
   ```

```
      "namespace": "<namespace>", 2
      "default-route": ["<default-route>"] 3
    }
  ]
```

1. NetworkAttachmentDefinition CR 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 CR 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 を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

```
$ oc create -f <name>.yaml
```

3. オプション: アノテーションが Pod CR 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、以下のコマンドを入力します。

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

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

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

1 **k8s.v1.cni.cncf.io/networks-status** 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について説明

7.2.1.1. Pod 固有のアドレスおよびルーティングオプションの指定

Pod を追加のネットワークに割り当てる場合、特定の Pod でそのネットワークに関するその他のプロパティーを指定する必要がある場合があります。これにより、ルーティングの一部を変更することができ、静的 IP アドレスおよび MAC アドレスを指定できます。これを実行するには、JSON 形式のアノテーションを使用できます。

前提条件

- Pod が追加ネットワークと同じ namespace にあること。
- OpenShift コマンドラインインターフェース (**oc**) のインストール。
- クラスターにログインすること。

手順

アドレスおよび/またはルーティングオプションを指定する間に Pod を追加のネットワークに追加するには、以下の手順を実行します。

1. Pod リソース定義を編集します。既存の Pod を編集する場合は、以下のコマンドを実行してデフォルトエディターでその定義を編集します。**<name>** を、編集する Pod の名前に置き換えます。

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

2. Pod リソース定義で、**k8s.v1.cni.cncf.io/networks** パラメーターを Pod の **metadata** マッピングに追加します。**k8s.v1.cni.cncf.io/networks** は、追加のプロパティーを指定するだけでなく、NetworkAttachmentDefinition カスタムリソース (CR) 名を参照するオブジェクト一覧の JSON 文字列を受け入れます。

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

1 **<network>** を、以下の例にあるように JSON オブジェクトに置き換えます。一重引用符が必要です。

3. 以下の例では、アノテーションで **default-route** パラメーターを使用して、デフォルトルートを持つネットワーク割り当てを指定します。

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '
    {
      "name": "net1"
    },
```

```
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

1. **name** キーは、Pod に関連付ける追加ネットワークの名前です。
2. **default-route** キーは、ルーティングテーブルに他のルーティングテーブルがない場合に、ルーティングされるトラフィックに使用されるゲートウェイ値を指定します。複数の **default-route** キーを指定すると、Pod がアクティブでなくなります。

デフォルトのルートにより、他のルートに指定されていないトラフィックがゲートウェイにルーティングされます。

重要

OpenShift Container Platform のデフォルトのネットワークインターフェース以外のインターフェースへのデフォルトのルートを設定すると、Pod 間のトラフィックについて予想されるトラフィックが別のインターフェースでルーティングされる可能性があります。

Pod のルーティングプロパティーを確認する場合、**oc** コマンドを Pod 内で **ip** コマンドを実行するために使用できます。

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

注記

また、Pod の **k8s.v1.cni.cncf.io/networks-status** を参照して、JSON 形式の一覧のオブジェクトで **default-route** キーの有無を確認し、デフォルトルートが割り当てられている追加ネットワークを確認することができます。

Pod に静的 IP アドレスまたは MAC アドレスを設定するには、JSON 形式のアノテーションを使用できます。これには、この機能をとくに許可するネットワークを作成する必要があります。これは、CNO の rawCNIConfig で指定できます。

1. 以下のコマンドを実行して CNO CR を編集します。

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

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
```

```
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、macvlan CNI プラグインを使用して静的 MAC アドレスと IP アドレスを使用するための設定パラメーターについて説明しています。

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

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

1. **plugins** フィールドは CNI 設定の設定一覧を指定します。
2. **capabilities** キーは、CNI プラグインのランタイム設定の静的 IP 機能を有効にするために要求が行われていることを示します。
3. **master** フィールドは macvlan プラグインに固有のフィールドです。
4. ここで、**capabilities** キーは、CNI プラグインの静的 MAC アドレス機能を有効にするために要求が行われていることを示します。

上記のネットワーク割り当ては、特定の Pod に割り当てられる静的 IP アドレスと MAC アドレスを指定するキーと共に、JSON 形式のアノテーションで参照できます。

以下を実行して必要な Pod を編集します。

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

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

```
apiVersion: v1
```

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

1. 上記の **rawCNIConfig** を作成する際に、指定されるように **<name>** を使用します。
2. 必要な IP アドレスを指定します。
3. 必要な MAC アドレスを指定します。

注記

静的 IP アドレスおよび MAC アドレスを同時に使用することはできません。これらは個別に使用することも、一緒に使用することもできます。

追加のネットワークを持つ Pod の IP アドレスと MAC プロパティーを検証するには、**oc** コマンドを使用して Pod 内で ip コマンドを実行します。

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

7.3. 追加ネットワークからの POD の削除

クラスターユーザーとして、追加のネットワークから Pod を削除できます。

7.3.1. 追加ネットワークからの Pod の削除

追加のネットワークから Pod を削除できます。

前提条件

- クラスター用に追加のネットワークを設定している。
- 追加のネットワークが Pod に割り当てられている。
- OpenShift CLI (**oc**) のインストール。
- クラスターにログインすること。

手順

追加のネットワークから Pod を削除するには、以下の手順を実行します。

1. 以下のコマンドを実行して Pod リソース定義を編集します。**<name>** を、編集する Pod の名前に置き換えます。

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

2. 以下のアクションのいずれかを実行して、Pod から追加のネットワークを削除するように **annotations** マッピングを更新します。

   - Pod からすべての追加ネットワークを削除するには、以下の例にあるように Pod リソース定義から **k8s.v1.cni.cncf.io/networks** パラメーターを削除します。

     ```
     apiVersion: v1
     kind: Pod
     metadata:
       name: example-pod
       annotations: {}
     spec:
       containers:
       - name: example-pod
         command: ["/bin/bash", "-c", "sleep 2000000000000"]
         image: centos/tools
     ```

   - Pod から特定の追加ネットワークを削除するには、追加ネットワークの NetworkAttachmentDefinition の名前を削除して **k8s.v1.cni.cncf.io/networks** パラメーターを更新します。

3. オプション: 以下のコマンドを実行して、Pod が追加のネットワークに接続されていないことを確認します。**<name>** を Pod の名前に置き換えます。

   ```
   $ oc describe pod <name>
   ```

   以下の例では、**example-pod** Pod はデフォルトのクラスターネットワークのみに割り当てられています。

   ```
   $ oc describe pod example-pod
   ```

   出力例

   ```
   Name:         example-pod
   ...
   Annotations:  k8s.v1.cni.cncf.io/networks-status:
                 [{
                     "name": "openshift-sdn",
                     "interface": "eth0",
                     "ips": [
                         "10.131.0.13"
                     ],
                     "default": true, (1)
                     "dns": {}
                 }]
   Status:       Running
   ...
   ```

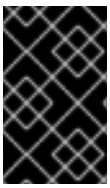
   (1) デフォルトのクラスターネットワークのみが Pod に割り当てられます。

7.4. ブリッジネットワークの設定

クラスター管理者は、ブリッジの Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。設定時に、ノード上のすべての Pod が仮想スイッチに接続されます。各 Pod には追加のネットワークの IP アドレスが割り当てられます。

7.4.1. ブリッジ CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は NetworkAttachmentDefinition カスタムリソース (CR) を自動的に作成します。



重要

Cluster Network Operator が管理する NetworkAttachmentDefinition CR は編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

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

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML はブリッジ CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "bridge",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "191.168.1.7"
          }
```

```
      ]
    }
  }'
```

1 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition CR を作成していることを確認します。CNO が CR を作成するまでに遅延が生じる可能性があります。

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

出力例

```
NAME             AGE
test-network-1    14m
```

7.4.1.1. ブリッジの設定

ブリッジ Container Network Interface (CNI) プラグインを使用する追加ネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

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

1 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。

2 ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。

3 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、ブリッジ CNI プラグインの設定パラメーターについて説明しています。

ブリッジ CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "type": "bridge",
  "bridge": "<bridge>", (2)
  "ipam": { (3)
    ...
  },
  "ipMasq": false, (4)
  "isGateway": false, (5)
  "isDefaultGateway": false, (6)
  "forceAddress": false, (7)
  "hairpinMode": false, (8)
  "promiscMode": false, (9)
  "vlan": <vlan>, (10)
  "mtu": <mtu> (11)
}
```

(1) CNO 設定に以前に指定した **name** パラメーターの値を指定します。

(2) 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ー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。

(3) IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、ネットワーク割り当て定義についての IP アドレスの割り当てを管理します。

(4) 仮想ネットワークから外すトラフィックについて 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ソース 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。

(5) 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。

(6) 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。

(7) 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。

(8) 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。

(9) 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。

(10) 仮想 LAN (VLAN) タグを整数値として指定します。デフォルトで、VLAN タグは割り当てません。

(11) 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。

7.4.1.1.1. ブリッジ設定の例

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

```
name: bridge-net
namespace: work-network
type: Raw
rawCNIConfig: '{ (1)
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}'
```

(1) CNI 設定オブジェクトは YAML 文字列として指定されます。

7.4.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。DHCP を使用して、静的 IP アドレスの割り当てまたは動的 IP アドレスの割り当てのいずれかに IPAM を設定することができます。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。

以下の JSON 設定オブジェクトは設定できるパラメーターについて説明しています。

7.4.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ (1)
      {
        "address": "<address>", (2)
        "gateway": "<gateway>" (3)
      }
    ],
    "routes": [ (4)
      {
        "dst": "<dst>" (5)
        "gw": "<gw>" (6)
      }
    ],
    "dns": { (7)
      "nameservers": ["<nameserver>"], (8)
      "domain": "<domain>", (9)
```

```
      "search": ["<search_domain>"] 10
    }
  }
}
```

1. 仮想インターフェースに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレス。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

7.4.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    rawCNIConfig: |-
    {
      "name": "dhcp-shim",
      "cniVersion": "0.3.1",
      "type": "bridge",
      "master": "ens5",
      "ipam": {
        "type": "dhcp"
      }
    }
```

DHCP 割り当ての設定

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

7.4.1.2.3. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

7.4.1.2.4. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

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

7.4.2. 次のステップ

- 追加ネットワークに Pod を割り当てます。

7.5. MACVLAN ネットワークの設定

クラスター管理者は、macvlan CNI プラグインを使用して、クラスターの追加のネットワークを設定できます。Pod がネットワークに割り当てられている場合、プラグインはホストの親インターフェースからサブインターフェースを作成します。各サブデバイスに対して固有のハードウェアの MAC アドレスが生成されます。

重要

このプラグインがサブインターフェース用に生成する固有の MAC アドレスは、クラウドプロバイダーのセキュリティーポリシーとの互換性がない場合があります。

7.5.1. macvlan CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は NetworkAttachmentDefinition カスタムリソース (CR) を自動的に作成します。

重要

Cluster Network Operator が管理する NetworkAttachmentDefinition CR は編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

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

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。

以下の YAML は、macvlan CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: SimpleMacvlan
    simpleMacvlanConfig:
      ipamConfig:
        type: static
        staticIPAMConfig:
          addresses:
          - address: 10.1.1.7
```

(1) 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition CR を作成していることを確認します。CNO が CR を作成するまでに遅延が生じる可能性があります。

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

出力例

```
NAME             AGE
test-network-1    14m
```

7.5.1.1. macvlan CNI プラグインの設定

以下の YAML は、macvlan Container Network Interface (CNI) プラグインの設定パラメーターについて説明しています。

macvlan YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
type: SimpleMacvlan
simpleMacvlanConfig:
  master: <master> (3)
  mode: <mode> (4)
  mtu: <mtu> (5)
  ipamConfig: (6)
    ...
```

(1) 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。

2 ネットワークの割り当てを作成する namespace を指定します。値が指定されない場合は、**default** namespace が使用されます。

3 仮想インターフェースに関連付けるイーサネットインターフェース。**master** の値が指定されない場合、ホストシステムのプライマリーイーサネットインターフェースが使用されます。

4 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。**mode** の値が指定されない場合、デフォルトの値は **bridge** になります。

5 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。

6 IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

7.5.1.1.1. macvlan 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。

```
name: macvlan-net
namespace: work-network
type: SimpleMacvlan
simpleMacvlanConfig:
  ipamConfig:
    type: DHCP
```

7.5.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。DHCP を使用して、静的 IP アドレスの割り当てまたは動的 IP アドレスの割り当てのいずれかに IPAM を設定することができます。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。

以下の YAML 設定は設定可能なパラメーターについて説明しています。

IPAM CNI プラグイン YAML 設定オブジェクト

```
ipamConfig:
  type: <type> 1
  ... 2
```

1 IP アドレスの割り当てを管理できるようにプラグインを設定するには **static** を指定します。**DHCP** を指定して、DHCP サーバーが IP アドレスの割り当てを管理できるようにします。**DHCP** の値を指定する場合は、追加のパラメーターを指定できません。

2 **type** パラメーターを **static** に設定する場合、**staticIPAMConfig** パラメーターを指定します。

7.5.1.2.1. 静的 IPAM 設定 YAML

以下の YAML は、静的 IP アドレスの割り当ての設定について説明しています。

静的 IPAM 設定 YAML

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses: 1
    - address: <address> 2
      gateway: <gateway> 3
    routes: 4
    - destination: <destination> 5
      gateway: <gateway> 6
    dns: 7
      nameservers: 8
      - <nameserver>
      domain: <domain> 9
      search: 10
      - <search_domain>
```

1. 仮想インターフェースに割り当てる IP アドレスを定義するマッピングのコレクション。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレス。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述するマッピングのコレクション。
5. CIDR 形式の IP アドレス範囲。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーを送信する 1 つ以上の IP アドレスのコレクション。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

7.5.1.2.2. 動的 IPAM 設定 YAML

以下の YAML は、静的 IP アドレスの割り当ての設定について説明しています。

動的 IPAM 設定 YAML

```
ipamConfig:
  type: DHCP
```

7.5.1.2.3. 静的 IP アドレス割り当ての設定例

以下の例は、静的 IP アドレスの IPAM 設定を示しています。

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses:
    - address: 10.51.100.11
      gateway: 10.51.100.10
    routes:
    - destination: 0.0.0.0/0
      gateway: 10.51.100.1
    dns:
      nameservers:
      - 10.51.100.1
      - 10.51.100.2
      domain: testDNS.example
      search:
      - testdomain1.example
      - testdomain2.example
```

7.5.1.2.4. 動的 IP アドレス割り当ての設定例

以下の例では、DHCP の IPAM 設定を示しています。

```
ipamConfig:
  type: DHCP
```

7.5.2. 次のステップ

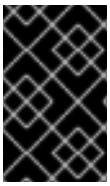
- 追加ネットワークに Pod を割り当てます。

7.6. IPVLAN ネットワークの設定

クラスター管理者は、ipvlan Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。このプラグインにより作成される仮想ネットワークは、指定する物理インターフェースに関連付けられます。

7.6.1. IPVLAN CNI プラグインを使用した追加のネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は NetworkAttachmentDefinition カスタムリソース (CR) を自動的に作成します。



重要

Cluster Network Operator が管理する NetworkAttachmentDefinition CR は編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールします。

- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

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

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML は IPVLAN CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "ipvlan",
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "191.168.1.7"
          }
        ]
      }
    }'
```

(1) 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. オプション: 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition CR を作成していることを確認します。CNO が CR を作成するまでに遅延が生じる可能性があります。

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

出力例

```
NAME             AGE
test-network-1   14m
```

7.6.1.1. IPVLAN の設定

IPVLAN Container Network Interface (CNI) プラグインを使用する追加のネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

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

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

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について説明しています。

IPVLAN CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "type": "ipvlan",
  "mode": "<mode>", 2
  "master": "<master>", 3
  "mtu": <mtu>, 4
  "ipam": { 5
    ...
  }
}
```

1. CNO 設定に以前に指定した **name** パラメーターの値を指定します。
2. 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。
3. 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ースを指定します。**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ースが使用されます。

4 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。

5 IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

7.6.1.1.1. IPVLAN 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。

```
name: ipvlan-net
namespace: work-network
type: Raw
rawCNIConfig: '{ 1
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "dhcp"
    }
}'
```

1 CNI 設定オブジェクトは YAML 文字列として指定されます。

7.6.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。DHCP を使用して、静的 IP アドレスの割り当てまたは動的 IP アドレスの割り当てのいずれかに IPAM を設定することができます。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。

以下の JSON 設定オブジェクトは設定できるパラメーターについて説明しています。

7.6.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

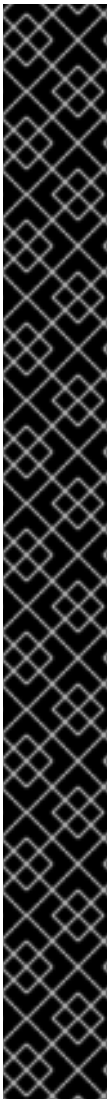
```
{
  "ipam": {
    "type": "static",
    "addresses": [ 1
      {
        "address": "<address>", 2
        "gateway": "<gateway>" 3
      }
    ],
    "routes": [ 4
```

```
      {
        "dst": "<dst>" 5
        "gw": "<gw>" 6
      }
    ],
    "dns": { 7
      "nameservers": ["<nameserver>"], 8
      "domain": "<domain>", 9
      "search": ["<search_domain>"] 10
    }
  }
}
```

1. 仮想インターフェースに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレス。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

7.6.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。



DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    rawCNIConfig: |-
    {
      "name": "dhcp-shim",
      "cniVersion": "0.3.1",
      "type": "bridge",
      "master": "ens5",
      "ipam": {
        "type": "dhcp"
      }
    }
```

DHCP 割り当ての設定

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

7.6.1.2.3. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

7.6.1.2.4. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

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

7.6.2. 次のステップ

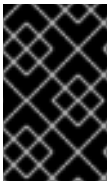
- 追加ネットワークに Pod を割り当てます。

7.7. ホストデバイスネットワークの設定

クラスター管理者は、ホストデバイス Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。このプラグインは、指定されたネットワークデバイスを、ホストのネットワーク namespace から Pod のネットワーク namespace に移動することを可能にします。

7.7.1. ホストデバイス CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は NetworkAttachmentDefinition カスタムリソース (CR) を自動的に作成します。



重要

Cluster Network Operator が管理する NetworkAttachmentDefinition CR は編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

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

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML は、ホストデバイス CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
```

```
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "host-device",
      "device": "eth1"
    }'
```

(1) 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition CR を作成していることを確認します。CNO が CR を作成するまでに遅延が生じる可能性があります。

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

出力例

```
NAME             AGE
test-network-1   14m
```

7.7.1.1. ホストデバイスの設定

ホストデバイス Container Network Interface (CNI) プラグインを使用する追加ネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

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

(1) 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。

2 ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。

3 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

重要

device、**hwaddr**、 **kernelpath**、または **pciBusID** のいずれかのパラメーターを設定してネットワークデバイスを指定します。

以下のオブジェクトは、ホストデバイス CNI プラグインの設定パラメーターについて説明しています。

ホストデバイス CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "type": "host-device",
  "device": "<device>", 2
  "hwaddr": "<hwaddr>", 3
  "kernelpath": "<kernelpath>", 4
  "pciBusID": "<pciBusID>", 5
    "ipam": { 6
     ...
  }
}
```

1 CNO 設定に以前に指定した **name** パラメーターの値を指定します。

2 **eth0**などのデバイスの名前を指定します。

3 デバイスハードウェアの MAC アドレスを指定します。

4 **/sys/devices/pci0000:00/0000:00:1f.6** などの Linux カーネルデバイスを指定します。

5 **0000:00:1f.6** などのネットワークデバイスの PCI アドレスを指定します。

6 IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

7.7.1.1.1. ホストデバイス設定例

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

```
name: hostdev-net
namespace: work-network
type: Raw
rawCNIConfig: '{ 1
  "cniVersion": "0.3.1",
  "name": "work-network",
```

```
  "type": "host-device",
  "device": "eth1"
}'
```

1 CNI 設定オブジェクトは YAML 文字列として指定されます。

7.7.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。DHCP を使用して、静的 IP アドレスの割り当てまたは動的 IP アドレスの割り当てのいずれかに IPAM を設定することができます。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。

以下の JSON 設定オブジェクトは設定できるパラメーターについて説明しています。

7.7.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ 1
      {
        "address": "<address>", 2
        "gateway": "<gateway>" 3
      }
    ],
    "routes": [ 4
      {
        "dst": "<dst>" 5
        "gw": "<gw>" 6
      }
    ],
    "dns": { 7
      "nameservers": ["<nameserver>"], 8
      "domain": "<domain>", 9
      "search": ["<search_domain>"] 10
    }
  }
}
```

1 仮想インターフェースに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。

2 指定する IP アドレス。

3 egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

4 Pod 内で設定するルートを記述する配列。

5 CIDR 形式の IP アドレス範囲。

6 ネットワークトラフィックがルーティングされるゲートウェイ。

7 オプション: DNS 設定。

8 DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。

9 ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。

10 DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

7.7.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    rawCNIConfig: |-
    {
      "name": "dhcp-shim",
      "cniVersion": "0.3.1",
      "type": "bridge",
      "master": "ens5",
      "ipam": {
        "type": "dhcp"
      }
    }
```

DHCP 割り当ての設定

```
{
  "ipam": {
```

```
    "type": "dhcp"
  }
}
```

7.7.1.2.3. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

7.7.1.2.4. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

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

7.7.2. 次のステップ

- 追加ネットワークに Pod を割り当てます。

7.8. 追加ネットワークの編集

クラスター管理者は、既存の追加ネットワークの設定を変更することができます。

7.8.1. 追加ネットワーク割り当て定義の変更

クラスター管理者は、既存の追加ネットワークに変更を加えることができます。追加ネットワークに割り当てられる既存の Pod は更新されません。

前提条件

- クラスター用に追加のネットワークを設定している。
- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターの追加ネットワークを編集するには、以下の手順を実行します。

1. 以下のコマンドを実行し、デフォルトのテキストエディターで Cluster Network Operator (CNO) CR を編集します。

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

2. **additionalNetworks** コレクションで、追加ネットワークを変更内容で更新します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. オプション: 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition CR を更新していることを確認します。**<network-name>** を表示する追加ネットワークの名前に置き換えます。CNO が NetworkAttachmentDefinition CR を更新して変更内容が反映されるまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

たとえば、以下のコンソールの出力は **net1** という名前の NetworkAttachmentDefinition を表示します。

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
{ "cniVersion": "0.3.1", "type": "macvlan",
"master": "ens5",
"mode": "bridge",
"ipam":       {"type":"static","routes":[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}],"dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}} }
```

7.9. 追加ネットワークの削除

クラスター管理者は、追加のネットワーク割り当てを削除できます。

7.9.1. 追加ネットワーク割り当て定義の削除

クラスター管理者は、追加ネットワークを OpenShift Container Platform クラスターから削除できます。追加ネットワークは、割り当てられている Pod から削除されません。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

クラスターから追加ネットワークを削除するには、以下の手順を実行します。

1. 以下のコマンドを実行して、デフォルトのテキストエディターで Cluster Network Operator (CNO) を編集します。

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

2. 削除しているネットワーク割り当て定義の **additionalNetworks** コレクションから設定を削除し、CR を変更します。

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

1 **additionalNetworks** コレクションの追加ネットワーク割り当てのみの設定マッピングを削除する場合、空のコレクションを指定する必要があります。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、追加ネットワーク CR が削除されていることを確認します。

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

7.10. PTP の設定

重要

Precision Time Protocol (PTP) ハードウェアはテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。

7.10.1. OpenShift Container Platform の PTP ハードウェアについて

OpenShift Container Platform には、ノード上で PTP ハードウェアを使用する機能が含まれます。linuxptp サービスは、PTP 対応ハードウェアを搭載したノードで設定できます。

PTP Operator をデプロイし、OpenShift Container Platform コンソールを使用して PTP をインストールできます。PTP Operator は、linuxptp サービスを作成し、管理します。Operator は以下の機能を提供します。

- クラスター内の PTP 対応デバイスを検出します。
- linuxptp サービスの設定を管理します。

7.10.2. PTP Operator のインストール

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して PTP Operator をインストールできます。

7.10.2.1. CLI を使用した Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- PTP に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

1. 以下のアクションを実行して、PTP Operator の namespace を作成します。
   a. **openshift-ptp** namespace を定義する以下の Namespace カスタムリソース (CR) を作成し、YAML を **ptp-namespace.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  labels:
    openshift.io/run-level: "1"
```

   b. 以下のコマンドを実行して namespace を作成します。

```
$ oc create -f ptp-namespace.yaml
```

2. 以下のオブジェクトを作成して、直前の手順で作成した namespace に PTP Operator をインストールします。
   a. 以下の OperatorGroup CR を作成し、YAML を **ptp-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
```

   b. 以下のコマンドを実行して OperatorGroup CR を作成します。

```
$ oc create -f ptp-operatorgroup.yaml
```

   c. 以下のコマンドを実行して、次の手順に必要な **channel** の値を取得します。

```
$ oc get packagemanifest ptp-operator -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'

4.3
```

d. 以下の Subscription CR を作成し、YAML を **ptp-sub.yaml** ファイルに保存します。

Subscription の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: <channel> 1
  name: ptp-operator
  source: redhat-operators 2
  sourceNamespace: openshift-marketplace
```

1 **.status.defaultChannel** パラメーターの直前の手順で取得した値を指定します。

2 **redhat-operators** 値を指定する必要があります。

e. 以下のコマンドを実行して Subscription オブジェクトを作成します。

```
$ oc create -f ptp-sub.yaml
```

f. **openshift-ptp** プロジェクトに切り替えます。

```
$ oc project openshift-ptp
```

出力例

```
Now using project "openshift-ptp"
```

7.10.2.2. Web コンソールでの Operator のインストール

クラスター管理者は、Web コンソールを使用して Operator をインストールできます。

注記

先のセクションで説明されているように Namespace CR および OperatorGroup CR を作成する必要があります。

手順

1. OpenShift Container Platform Web コンソールを使用して PTP Operator をインストールします。

   a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。

   b. 利用可能な Operator の一覧から **PTP Operator** を選択してから **Install** をクリックします。

c. **Create Operator Subscription** ページの **A specific namespace on the cluster**の下で **openshift-ptp** を選択します。次に、**Subscribe** をクリックします。

2. オプション: PTP Operator が正常にインストールされていることを確認します。

a. **Operators → Installed Operators** ページに切り替えます。

b. **PTP Operator** が **Status** が **InstallSucceeded** の状態で **openshift-ptp** プロジェクトに一覧表示されていることを確認します。

注記

インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operators → Installed Operators** ページに移動し、**Operator Subscriptions** および **Install Plans** タブで **Status** の下にエラーがあるかどうかを検査します。
- **Workloads → Pods** ページに移動し、**openshift-ptp** プロジェクトで Pod のログを確認します。

7.10.3. PTP ネットワークデバイスの自動検出

PTP Operator は **NodePtpDevice.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。PTP Operator はクラスターで、各ノードの PTP 対応ネットワークデバイスを検索します。Operator は、互換性のある PTP デバイスを提供する各ノードの **NodePtpDevice** カスタムリソース (CR) を作成し、更新します。

1 つの CR がノードごとに作成され、ノードと同じ名前を共有します。**.status.devices** 一覧は、ノード上の PTP デバイスについての情報を提供します。

以下は、PTP Operator によって作成される NodePtpDevice CR の例です。

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
  generation: 1
  name: dev-worker-0 (1)
  namespace: openshift-ptp (2)
  resourceVersion: "487462"
  selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-ptp/nodeptpdevices/dev-worker-0
  uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
spec: {}
status:
  devices: (3)
  - name: eno1
  - name: eno2
  - name: ens787f0
  - name: ens787f1
  - name: ens801f0
```

```
- name: ens801f1
- name: ens802f0
- name: ens802f1
- name: ens803
```

1. **name** パラメーターの値はノードの名前と同じです。
2. CR は PTP Operator によって **openshift-ptp** namespace に作成されます。
3. **devices** コレクションには、ノード上の Operator によって検出されるすべての PTP 対応デバイスの一覧が含まれます。

7.10.4. Linuxptp サービスの設定

PTP Operator は **PtpConfig.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。**PtpConfig** カスタムリソース (CR) を作成して、Linuxptp サービス (ptp4l、phc2sys) を設定できます。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログイン。
- PTP Operator がインストールされていること。

手順

1. 以下の **PtpConfig** CR を作成してから、YAML を **<name>-ptp-config.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <name> 1
  namespace: openshift-ptp 2
spec:
  profile: 3
  - name: "profile1" 4
    interface: "ens787f1" 5
    ptp4lOpts: "-s -2" 6
    phc2sysOpts: "-a -r" 7
  recommend: 8
  - profile: "profile1" 9
    priority: 10 10
    match: 11
    - nodeLabel: "node-role.kubernetes.io/worker" 12
      nodeName: "dev-worker-0" 13
```

1. **PtpConfig** CR の名前を指定します。
2. PTP Operator がインストールされている namespace を指定します。

3 1つ以上の **profile** オブジェクトの配列を指定します。

4 プロファイルオブジェクトを一意に識別するために使用されるプロファイルオブジェクトの名前を指定します。

5 **ptp4l** サービスで使用するネットワークインターフェース名を指定します (例: **ens787f1**)。

6 **ptp4l** サービスのシステム設定オプション (例: **-s -2**) を指定します。これには、インターフェース名 **-i <interface>** およびサービス設定ファイル **-f /etc/ptp4l.conf** を含めないでください。これらは自動的に追加されます。

7 **phc2sys** サービスのシステム設定オプション（例: **-a -r**) を指定します。

8 **profile** がノードに適用される方法を定義する1つ以上の **recommend** オブジェクトの配列を指定します。

9 **profile** セクションに定義される **profile** オブジェクト名を指定します。

10 **0** から **99** までの整数値で **priority** を指定します。数値が大きいほど優先度が低くなるため、**99** の優先度は **10**よりも低くなります。ノードが **match** フィールドで定義されるルールに基づいて複数のプロファイルに一致する場合、優先順位の高い プロファイルがそのノードに適用されます。

11 **match** ルールを、**nodeLabel** または **nodeName** で指定します。

12 **nodeLabel** を、ノードオブジェクトの **node.Labels** の **key** で指定します。

13 **nodeName** をノードオブジェクトの **node.Name** で指定します。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f <filename> 1
```

1 **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. オプション: **PtpConfig** プロファイルが、 **nodeLabel** または **nodeName** に一致するノードに適用されることを確認します。

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

出力例

```
NAME                           READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
linuxptp-daemon-4xkbb          1/1     Running   0          43m   192.168.111.15   dev-worker-0
<none>           <none>
linuxptp-daemon-tdspf          1/1     Running   0          43m   192.168.111.11   dev-master-0
<none>           <none>
ptp-operator-657bbb64c8-2f8sj  1/1     Running   0          43m   10.128.0.116     dev-master-0
<none>           <none>

$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
```

```
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1 1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1 2
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -s -2 3
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r 4
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
I1115 09:41:18.117934 4143292 daemon.go:186] Starting phc2sys...
I1115 09:41:18.117985 4143292 daemon.go:187] phc2sys cmd: &{Path:/usr/sbin/phc2sys
Args:[/usr/sbin/phc2sys -a -r] Env:[] Dir: Stdin:<nil> Stdout:<nil> Stderr:<nil> ExtraFiles:[]
SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil> lookPathErr:<nil> finished:false
childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[] errch:<nil> waitDone:<nil>}
I1115 09:41:19.118175 4143292 daemon.go:186] Starting ptp4l...
I1115 09:41:19.118209 4143292 daemon.go:187] ptp4l cmd: &{Path:/usr/sbin/ptp4l Args:
[/usr/sbin/ptp4l -m -f /etc/ptp4l.conf -i ens787f1 -s -2] Env:[] Dir: Stdin:<nil> Stdout:<nil>
Stderr:<nil> ExtraFiles:[] SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil>
lookPathErr:<nil> finished:false childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[]
errch:<nil> waitDone:<nil>}
ptp4l[102189.864]: selected /dev/ptp5 as PTP clock
ptp4l[102189.886]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[102189.886]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
```

1. **Profile Name** は、ノード **dev-worker-0** に適用される名前です。
2. **Interface** は、**profile1** インターフェースフィールドに指定される PTP デバイスです。**ptp4l** サービスはこのインターフェースで実行されます。
3. **PTP4lOpts** は、**profile1** Ptp4lOpts フィールドで指定される ptp4l sysconfig オプションです。
4. **phc2sysOpts** は、**profile1** phc2sysOpts フィールドで指定される phc2sys sysconfig オプションです。

第8章 ハードウェアネットワーク

8.1. SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて

Single Root I/O Virtualization (SR-IOV) 仕様は、単一デバイスを複数の Pod で共有できる PCI デバイス割り当てタイプの標準です。

SR-IOV を使用すると、準拠したネットワークデバイス (ホストノードで物理機能 (PF) として認識される) を複数の仮想機能 (VF) にセグメント化することができます。VF は他のネットワークデバイスと同様に使用されます。デバイスの SR-IOV デバイスドライバーは、VF がコンテナーで公開される方法を判別します。

- **netdevice** ドライバー: コンテナーの **netns** 内の通常のカーネルネットワークデバイス
- **vfio-pci** ドライバー: コンテナーにマウントされるキャラクターデバイス

高帯域幅または低レイテンシーを必要とするアプリケーション用に、OpenShift Container Platform クラスターの追加のネットワークと共に SR-IOV ネットワークデバイスを使用できます。

8.1.1. SR-IOV ネットワークデバイスを管理するコンポーネント

SR-IOV ネットワーク Operator は SR-IOV スタックのコンポーネントを作成し、管理します。以下の機能を実行します。

- SR-IOV ネットワークデバイスの検出および管理のオーケストレーション
- SR-IOV Container Network Interface (CNI) の NetworkAttachmentDefinition カスタムリソースの生成
- SR-IOV ネットワークデバイスプラグインの設定の作成および更新
- ノード固有の SriovNetworkNodeState カスタムリソースの作成
- 各 SriovNetworkNodeState カスタムリソースの **spec.interfaces** フィールドの更新

Operator は以下のコンポーネントをプロビジョニングします。

SR-IOV ネットワーク設定デーモン

SR-IOV Operator の起動時にワーカーノードにデプロイされる DaemonSet。デーモンは、クラスターで SR-IOV ネットワークデバイスを検出し、初期化します。

SR-IOV Operator Webhook

Operator カスタムリソースを検証し、未設定フィールドに適切なデフォルト値を設定する動的受付コントローラー Webhook。

SR-IOV Network Resources Injector

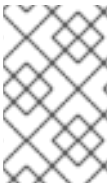
SR-IOV VF などのカスタムネットワークリソースの要求および制限のある Kubernetes Pod 仕様のパッチを適用するための機能を提供する動的受付コントローラー Webhook。

SR-IOV ネットワークデバイスプラグイン

SR-IOV ネットワーク仮想機能 (VF) リソースの検出、公開、割り当てを実行するデバイスプラグイン。デバイスプラグインは、とりわけ物理デバイスでの制限されたリソースの使用を有効にするために Kubernetes で使用されます。デバイスプラグインは Kubernetes スケジューラーにリソースの可用性を認識させるため、スケジューラーはリソースが十分にあるノードで Pod をスケジュールできます。

SR-IOV CNI プラグイン

SR-IOV デバイスプラグインから割り当てられる VF インターフェースを直接 Pod に割り当てる CNI プラグイン。



注記

SR-IOV Network Resources Injector および SR-IOV Network Operator Webhook は、デフォルトで有効にされ、**default** の SriovOperatorConfig CR を編集して無効にできます。

8.1.1.1. サポートされるデバイス

以下の Network Interface Card (NIC) モデルは、OpenShift Container Platform でサポートされています。

- Intel XXV710-DA2 25G カード (ベンダー ID **0x8086** およびデバイス ID **0x158b**)
- Mellanox MT27710 Family [ConnectX-4 Lx] 25G カード (ベンダー ID **0x15b3** およびデバイス ID **0x1015**)
- Mellanox MT27800 Family [ConnectX-5] 100G カード (ベンダー ID **0x15b3** およびデバイス ID **0x1017**)

8.1.1.2. Pod での 仮想機能 (VF) の使用例

SR-IOV VF が割り当てられている Pod で、Remote Direct Memory Access (RDMA) または Data Plane Development Kit (DPDK) アプリケーションを実行できます。

以下の例では、RDMA モードで仮想機能 (VF) を使用する Pod を示しています。

RDMA モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
     capabilities:
        add: ["IPC_LOCK"]
    command: ["sleep", "infinity"]
```

以下の例は、DPDK モードの VF のある Pod を示しています。

DPDK モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
     capabilities:
        add: ["IPC_LOCK"]
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
      requests:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

オプションのライブラリーは、コンテナーで実行されるアプリケーションによる Pod 関連のネットワーク情報を収集を支援するために利用できます。このライブラリーは 'app-netutil' と呼ばれます。**app-netutil** GitHub リポジトリー でライブラリーのソースコードを参照してください。

このライブラリーは、DPDK モードの SR-IOV VF のコンテナーへの統合を容易にすることを目的としています。ライブラリーは、GO API と C API、および両方の言語の使用例を提供します。

また、サンプルの Docker イメージ 'dpdk-app-centos' も用意されています。このイメージは、Pod 仕様の l2fwd、l3wd または testpmd の環境変数に基づいて、以下の DPDK サンプルアプリケーションのいずれかを実行できます。この Docker イメージは、「app-netutil」をコンテナーイメージ自体に統合するサンプルを提供します。ライブラリーも、必要なデータを収集し、データを既存の DPDK ワークロードに渡す init-container に統合できます。

8.1.2. 次のステップ

- SR-IOV ネットワーク Operator のインストール
- オプション: SR-IOV ネットワーク Operator の設定
- SR-IOV ネットワークデバイスの設定
- SR-IOV ネットワーク割り当ての設定
- Pod の SR-IOV の追加ネットワークへの追加

8.2. SR-IOV ネットワーク OPERATOR のインストール

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator をクラスターにインストールし、SR-IOV ネットワークデバイスとネットワークの割り当てを管理できます。

8.2.1. SR-IOV ネットワーク Operator のインストール

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して SR-IOV ネットワーク Operator をインストールできます。

8.2.1.1. CLI: SR-IOV ネットワーク Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** 権限を持つアカウント。

手順

1. **openshift-sriov-network-operator** namespace を作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  labels:
    openshift.io/run-level: "1"
EOF
```

2. OperatorGroup CR を作成するには、以下のコマンドを実行します。

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
EOF
```

3. SR-IOV ネットワーク Operator にサブスクライブします。

   a. 以下のコマンドを実行して OpenShift Container Platform のメジャーおよびマイナーバージョンを取得します。これは、次の手順の **channel** の値に必要です。

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

b. SR-IOV ネットワーク Operator の Subscription CR を作成するには、以下のコマンドを入力します。

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

4. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

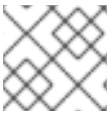
```
$ oc get csv -n openshift-sriov-network-operator \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                                         Phase
sriov-network-operator.4.4.0-202006160135   Succeeded
```

8.2.1.2. Web コンソール: SR-IOV ネットワーク Operator のインストール

クラスター管理者は、Web コンソールを使用して Operator をインストールできます。



注記

CLI を使用して OperatorGroup CR を作成する必要があります。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** 権限を持つアカウント。

手順

1. SR-IOV ネットワーク Operator の namespace を作成します。
   a. OpenShift Container Platform Web コンソールで、**Administration → Namespaces** をクリックします。

b. **Create Namespace** をクリックします。

c. **Name** フィールドに **openshift-sriov-network-operator** を入力し、**Create** をクリックします。

d. **Filter by name** フィールドに、**openshift-sriov-network-operator** を入力します。

e. 結果の一覧から **openshift-sriov-network-operator** をクリックした後、**YAML** をクリックします。

f. namespace 定義に以下のスタンザを追加して namespace を更新します。

```
labels:
  openshift.io/run-level: "1"
```

g. **Save** をクリックします。

2. SR-IOV ネットワーク Operator をインストールします。

a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。

b. 利用可能な Operator の一覧から **SR-IOV Network Operator** を選択してから **Install** をクリックします。

c. **Create Operator Subscription** ページの **A specific namespace on the cluster**の下で、**openshift-sriov-network-operator** を選択します。

d. **Subscribe** をクリックします。

3. SR-IOV ネットワーク Operator が正常にインストールされていることを確認します。

a. **Operators → Installed Operators** ページに移動します。

b. **Status** が **InstallSucceeded** の状態で、**SR-IOV Network Operator** が **openshift-sriov-network-operator** プロジェクトに一覧表示されていることを確認します。

注記

インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operator Subscriptions** および **Install Plans** タブで、**Status** の下の失敗またはエラーの有無を確認します。
- **Workloads → Pods** ページに移動し、**openshift-sriov-network-operator** プロジェクトで Pod のログを確認します。

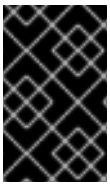
8.2.2. 次のステップ

- オプション: SR-IOV ネットワーク Operator の設定

8.3. SR-IOV ネットワーク OPERATOR の設定

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator は、クラスターで SR-IOV ネットワークデバイスおよびネットワーク割り当てを管理します。

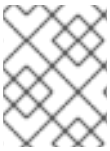
8.3.1. SR-IOV ネットワーク Operator の設定



重要

通常、SR-IOV ネットワーク Operator 設定を変更する必要はありません。デフォルト設定は、ほとんどのユースケースで推奨されます。Operator のデフォルト動作がユースケースと互換性がない場合にのみ、関連する設定を変更する手順を実行します。

SR-IOV ネットワーク Operator は **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition リソースを追加します。Operator は、**openshift-sriov-network-operator** namespace に **default** という名前の SriovOperatorConfig カスタムリソース (CR) を自動的に作成します。



注記

default CR には、クラスターの SR-IOV ネットワーク Operator 設定が含まれます。Operator 設定を変更するには、この CR を変更する必要があります。

SriovOperatorConfig CR は、Operator を設定するための複数のフィールドを提供します。

- **enableInjector** を使用すると、プロジェクト管理者は Network Resources Injector DaemonSet を有効または無効にすることができます。
- **enableOperatorWebhook** を使用すると、プロジェクト管理者は Operator Admission Controller webook DaemonSet を有効または無効にすることができます。
- **configDaemonNodeSelector** を使用すると、プロジェクト管理者は選択したノードで SR-IOV Network Config Daemon をスケジュールできます。

8.3.1.1. Network Resources Injector について

Network Resources Injector は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- SR-IOV リソース名を SR-IOV ネットワーク割り当て定義アノテーションに従って追加するための、Pod 仕様でのリソース要求および制限の変更。
- Pod のアノテーションおよびラベルを **/etc/podnetinfo** パスの下にあるファイルとして公開するための、Downward API ボリュームでの Pod 仕様の変更。

デフォルトで、Network Resources Injector は SR-IOV Operator によって有効にされ、すべてのマスターノードで DaemonSet として実行されます。以下は、3 つのマスターノードを持つクラスターで実行される Network Resources Injector Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

```
NAME                                     READY   STATUS    RESTARTS   AGE
network-resources-injector-5cz5p         1/1     Running   0          10m
network-resources-injector-dwqpx         1/1     Running   0          10m
network-resources-injector-lktz5         1/1     Running   0          10m
```

8.3.1.2. SR-IOV Operator Admission Controller Webhook について

SR-IOV Operator Admission Controller Webook は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- 作成時または更新時の **SriovNetworkNodePolicy** CR の検証
- CR の作成または更新時の **priority** および **deviceType** フィールドのデフォルト値の設定による **SriovNetworkNodePolicy** CR の変更

デフォルトで、SR-IOV Operator Admission Controller Webook は Operator によって有効にされ、すべてのマスターノードで DaemonSet として実行されます。以下は、3 つのマスターノードを持つクラスターで実行される Operator Admission Controller Webook Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

```
NAME                                     READY   STATUS    RESTARTS   AGE
operator-webhook-9jkw6                   1/1     Running   0          16m
operator-webhook-kbr5p                   1/1     Running   0          16m
operator-webhook-rpfrl                   1/1     Running   0          16m
```

8.3.1.3. カスタムノードセレクターについて

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

8.3.1.4. Network Resources Injector の無効化または有効化

デフォルトで有効にされている Network Resources Injector を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログイン。
- SR-IOV Operator がインストールされていること。

手順

- **enableInjector** フィールドを設定します。**<value>** を **false** に置き換えて機能を無効にするか、または **true** に置き換えて機能を有効にします。

```
$ oc patch sriovoperatorconfig default \
  --type=merge -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableInjector": <value> } }'
```

8.3.1.5. SR-IOV Operator Admission Controller Webhook の無効化または有効化

デフォルトで有効にされている なっている受付コントローラー Webhook を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログイン。
- SR-IOV Operator がインストールされていること。

手順

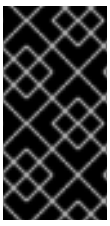
- **enableOperatorWebhook** フィールドを設定します。**<value>** を **false** に置き換えて機能を無効するか、**true** に置き換えて機能を有効にします。

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

8.3.1.6. SRIOV Network Config Daemon のカスタム NodeSelector の設定

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

SR-IOV Network Config デーモンがデプロイされるノードを指定するには、以下の手順を実行します。



重要

configDaemonNodeSelector フィールドを更新する際に、SR-IOV Network Config デーモンがそれぞれの選択されたノードに再作成されます。デーモンが再作成されている間、クラスターのユーザーは新規の SR-IOV Network ノードポリシーを適用したり、新規の SR-IOV Pod を作成したりできません。

手順

- Operator のノードセレクターを更新するには、以下のコマンドを入力します。

```
$ oc patch sriovoperatorconfig default --type=json \
  -n openshift-sriov-network-operator \
  --patch '[{
      "op": "replace",
      "path": "/spec/configDaemonNodeSelector",
      "value": {<node-label>}
    }]'
```

以下の例のように、**<node-label>** を適用するラベルに置き換えます: **"node-role.kubernetes.io/worker": ""**.

8.3.2. 次のステップ

- SR-IOV ネットワークデバイスの設定

8.4. SR-IOV ネットワークデバイスの設定

クラスターで Single Root I/O Virtualization (SR-IOV) デバイスを設定できます。

8.4.1. SR-IOV ネットワークデバイスの自動検出

SR-IOV ネットワーク Operator は、クラスターでワーカーノード上の SR-IOV 対応ネットワークデバイスを検索します。Operator は、互換性のある SR-IOV ネットワークデバイスを提供する各ワーカーノードの SriovNetworkNodeState カスタムリソース (CR) を作成し、更新します。

CR にはワーカーノードと同じ名前が割り当てられます。**status.interfaces** 一覧は、ノード上のネットワークデバイスについての情報を提供します。

重要

SriovNetworkNodeState CR は変更しないでください。Operator はこれらのリソースを自動的に作成し、管理します。

8.4.1.1. SriovNetworkNodeState CR の例

以下の YAML は、SR-IOV ネットワーク Operator によって作成される SriovNetworkNodeState CR の例です。

SriovNetworkNodeState オブジェクト

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 (1)
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: (2)
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
```

```
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
syncStatus: Succeeded
```

1. **name** フィールドの値はワーカーノードの名前と同じです。
2. **interfaces** スタンザには、ワーカーノード上の Operator によって検出されるすべての SR-IOV デバイスの一覧が含まれます。

8.4.2. SR-IOV ネットワークデバイスの設定

SR-IOV ネットワーク Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、SriovNetworkNodePolicy カスタムリソース (CR) を作成して設定できます。

注記

SriovNetworkNodePolicy CR で指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分かかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つアカウント。

- SR-IOV Operator がインストールされていること。

手順

1. 以下の SriovNetworkNodePolicy CR を作成してから、YAML を **<name>-sriov-node-network.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: <device_type> 13
  isRdma: false 14
```

1. CR オブジェクトの名前を指定します。
2. SR-IOV Operator がインストールされている namespace を指定します。
3. SR-IOV デバイスプラグインのリソース名を指定します。1 つのリソース名に複数の SriovNetworkNodePolicy CR を作成できます。
4. 設定するノードを選択するノードセレクターを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイスプラグインは、選択したノードにのみデプロイされます。
5. オプション: **0** から **99** までの整数値を指定します。数値が小さいほど優先度が高くなります。したがって、**10** は **99** よりも優先度が高くなります。デフォルト値は **99** です。
6. オプション: 仮想機能 (VF) の最大転送単位 (MTU) の値を指定します。MTU の最大値は NIC モデルによって異なります。
7. SR-IOV 物理ネットワークデバイス用に作成する仮想機能 (VF) の数を指定します。Intel Network Interface Card (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **128** よりも大きくすることはできません。
8. **nicSelector** マッピングは、Operator が設定するイーサネットデバイスを選択します。すべてのパラメーターの値を指定する必要はありません。意図せずにイーサネットデバイスを選択する可能性を最低限に抑えるために、イーサネットアダプターを正確に特定できるようにすることが推奨されます。**rootDevices** を指定する場合、**vendor**、 **deviceID**、または **pfName** の値も指定する必要があります。**pfNames** と **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスをポイントすることを確認します。

9 オプション: SR-IOV ネットワークデバイスのベンダー 16 進コードを指定します。許可される値は **8086** または **15b3** のいずれかのみになります。

10 オプション: SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。許可される値は **158b**、**1015**、**1017** のみになります。

11 オプション: このパラメーターは、1 つ以上のイーサネットデバイスの物理機能 (PF) 名の配列を受け入れます。

12 このパラメーターは、イーサネットデバイスの物理機能についての 1 つ以上の PCI バスアドレスの配列を受け入れます。以下の形式でアドレスを指定します: **0000:02:00.1**

13 オプション: 仮想機能 (VF) のドライバータイプを指定します。以下の値のいずれかを指定できます: **netdevice** または **vfio-pci**デフォルト値は **netdevice** です。

注記

Mellanox カードをベアメタルノードの Data Plane Development Kit (DPDK) モードで機能させるには、**netdevice** ドライバータイプを使用し、**isRdma** を **true** に設定します。Mellanox カードを Container-Native Virtualization (CNV) を使用して DPDK モードで機能させるには、**vfio-pci** ドライバータイプを使用し、**isRdma** を **false** に設定します。

14 オプション。Remote Direct Memory Access (RDMA) モードを有効にするかどうかを指定します。デフォルト値は **false** です。Mellanox Ethernet アダプターでは、RoCE (RDMA over Converged Ethernet) モードのみがサポートされます。

注記

isRDMA フラグが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

2. SriovNetworkNodePolicy CR を作成します。**<name>** をこの設定の名前に置き換えます。

```
$ oc create -f <name>-sriov-node-network.yaml
```

設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

3. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

8.4.3. 次のステップ

- SR-IOV ネットワーク割り当ての設定

8.5. SR-IOV ネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスのネットワーク割り当てを設定できます。

8.5.1. SR-IOV の追加ネットワークの設定

SriovNetwork custom resource (CR) を作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。SriovNetwork CR の作成時に、SR-IOV Operator は NetworkAttachmentDefinition CR を自動的に作成します。

注記

SriovNetwork CR が **running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログインします。

手順

1. 以下の SriovNetwork CR を作成してから、YAML を **<name>-sriov-network.yaml** ファイルに保存します。**<name>** を、この追加ネットワークの名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  networkNamespace: <target_namespace> 3
  ipam: <ipam> 4
  vlan: <vlan> 5
  resourceName: <sriov_resource_name> 6
  linkState: <link_state> 7
  maxTxRate: <max_tx_rate> 8
  minTxRate: <min_rx_rate> 9
  vlanQoS: <vlan_qos> 10
  spoofChk: "<spoof_check>" 11
  trust: "<trust_vf>" 12
  capabilities: <capabilities> 13
```

1. **<name>** を CR の名前に置き換えます。SR-IOV ネットワーク Operator は、同じ名前を持つ NetworkAttachmentDefinition CR を作成します。
2. SR-IOV Operator がインストールされている namespace を指定します。
3. オプション: **<target_namespace>** を NetworkAttachmentDefinition CR が作成される namespace に置き換えます。デフォルト値は **openshift-sriov-network-operator** です。
4. オプション: **<ipam>** を YAML Block Scaler として IPAM CNI プラグインの設定オブジェクトに置き換えます。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

5 オプション: **<vlan>** を、追加ネットワークの仮想 LAN (VLAN) ID に置き換えます。整数値は **0** から **4095**である必要があります。デフォルト値は **0** です。

6 **<sriov_resource_name>** を、この追加ネットワークの SR-IOV ハードウェアを定義する SriovNetworkNodePolicy CR の **.spec.resourceName** パラメーターの値に置き換えます。

7 オプション: **<link_state>** を仮想機能 (VF) のリンクの状態に置き換えます。許可される値は、**enable**、**disable**、および **auto** です。

8 オプション: **<max_tx_rate>** を VF の最大伝送レート (Mbps) に置き換えます。

9 オプション: **<min_tx_rate>** を VF の最小伝送レート (Mbps) に置き換えます。この値は、常に最大伝送レート以下である必要があります。

注記

Intel NIC は **minTxRate** パラメーターをサポートしません。詳細は、BZ#1772847 を参照してください。

10 オプション: **<vlan_qos>** を VF の IEEE 802.1p 優先レベルに置き換えます。デフォルト値は **0** です。

11 オプション: **<spoof_check>** を VF の spoof check モードに置き換えます。許可される値は、文字列の **"on"** および **"off"** です。

重要

指定する値を引用符で囲む必要があります。そうしないと、CR は SR-IOV ネットワーク Operator によって拒否されます。

12 オプション: **<trust_vf>** を VF の信頼モードに置き換えます。許可される値は、文字列の **"on"** および **"off"** です。

重要

指定する値を引用符で囲む必要があります。そうしないと、CR は SR-IOV ネットワーク Operator によって拒否されます。

13 オプション: **<capabilities>** を、このネットワークに設定する機能に置き換えます。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

2. CR 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。**<name>** を、この追加ネットワークの名前に置き換えます。

```
$ oc create -f <name>-sriov-network.yaml
```

3. オプション: 以下のコマンドを実行して、直前の手順で作成した SriovNetwork CR に関連付けられた NetworkAttachmentDefinition CR が存在することを確認します。**<namespace>** を、SriovNetwork CR で指定した namespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

8.5.1.1. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。DHCP を使用して、静的 IP アドレスの割り当てまたは動的 IP アドレスの割り当てのいずれかに IPAM を設定することができます。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。

以下の JSON 設定オブジェクトは設定できるパラメーターについて説明しています。

8.5.1.1.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ 1
      {
        "address": "<address>", 2
        "gateway": "<gateway>" 3
      }
    ],
    "routes": [ 4
      {
        "dst": "<dst>" 5
        "gw": "<gw>" 6
      }
    ],
    "dns": { 7
      "nameservers": ["<nameserver>"], 8
      "domain": "<domain>", 9
      "search": ["<search_domain>"] 10
    }
  }
}
```

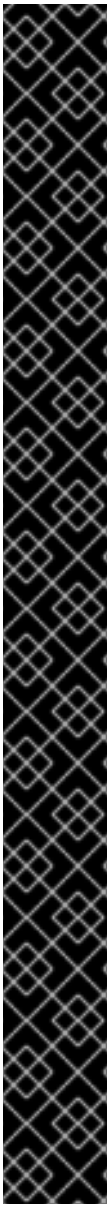
1. 仮想インターフェースに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレス。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。

9 ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き

10 DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

8.5.1.1.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。



DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

SR-IOV ネットワーク Operator は DHCP サーバーデプロイメントを作成しません。Cluster Network Operator は最小限の DHCP サーバーデプロイメントを作成します。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    rawCNIConfig: |-
    {
      "name": "dhcp-shim",
      "cniVersion": "0.3.1",
      "type": "bridge",
      "master": "ens5",
      "ipam": {
        "type": "dhcp"
      }
    }
```

DHCP 割り当ての設定

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

8.5.1.1.3. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

8.5.1.1.4. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

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

8.5.1.2. 追加の SR-IOV ネットワークでの静的 MAC および IP アドレスの設定

Pod アノテーションに Container Network Interface (CNI) runtimeConfig データを指定し、追加の SR-IOV ネットワークで静的 MAC および IP アドレスを設定できます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- SriovNetwork CR の作成時に **cluster-admin** 権限を持つユーザーとしてログインします。

手順

1. 以下の SriovNetwork CR を作成してから、YAML を **<name>-sriov-network.yaml** ファイルに保存します。**<name>** を、この追加ネットワークの名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  networkNamespace: <target_namespace> 3
  ipam: '{ "type": "static" }' 4
  capabilities: '{ "mac": true, "ips": true }' 5
  resourceName: <sriov_resource_name> 6
```

1. **<name>** を CR の名前に置き換えます。SR-IOV ネットワーク Operator は、同じ名前を持つ NetworkAttachmentDefinition CR を作成します。

2 SR-IOV ネットワーク Operator がインストールされている namespace を指定します。

3 **<target_namespace>** を NetworkAttachmentDefinition CR が作成される namespace に置き換えます。

4 IPAM CNI プラグインの静的タイプを YAML ブロックスケーラーとして指定します。

5 **mac** および **ips capabilities** を **true** に指定します。

6 **<sriov_resource_name>** を、この追加ネットワークの SR-IOV ハードウェアを定義する SriovNetworkNodePolicy CR の **spec.resourceName** パラメーターの値に置き換えます。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f <filename> 1
```

1 **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. オプション: 以下のコマンドを実行して、直前の手順で作成した SriovNetwork CR に関連付けられた NetworkAttachmentDefinition CR が存在することを確認します。**<namespace>** を、SriovNetwork CR で指定した namespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

注記

SriovNetwork Custom Resource (CR) が **running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

1. 以下の SR-IOV Pod 仕様を作成してから、YAML を **<name>-sriov-pod.yaml** ファイルに保存します。**<name>** をこの Pod の名前に置き換えます。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
 {
  "name": "<name>", 1
  "mac": "20:04:0f:f1:88:01", 2
  "ips": ["192.168.10.1/24", "2001::1/64"] 3
 }
]'
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

1 SR-IOV ネットワーク割り当て定義 CR の名前を指定します。

2 SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレスを指定します。

3 SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスのアドレスを指定します。IPv4 と IPv6 アドレスの両方がサポートされます。

2. 以下のコマンドを実行して SR-IOV Pod のサンプルを作成します。

```
$ oc create -f <filename> 1
```

1 **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. オプション: 以下のコマンドを実行して、**mac** と **ips** アドレスが SR-IOV デバイスに適用されていることを確認します。**<namespace>** を、SriovNetwork CR で指定した namespace に置き換えます。

```
$ oc exec sample-pod -n <namespace> -- ip addr show
```

8.5.2. 次のステップ

- Pod の SR-IOV の追加ネットワークへの追加

8.6. POD の SR-IOV の追加ネットワークへの追加

Pod を既存の Single Root I/O Virtualization (SR-IOV) ネットワークに追加できます。

8.6.1. 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 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。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当てることはできません。

注記

NetworkAttachmentDefinition が SR-IOV ネットワーク Operator によって管理される場合、SR-IOV Network Resource Injector は **resource** フィールドを Pod オブジェクトに自動的に追加します。

重要

Deployment リソースまたは ReplicationController リソースの SR-IOV ハードウェアネットワークを指定する場合、NetworkAttachmentDefinition CR の namespace を指定する必要があります。詳細は、以下の BZ#1846333 および BZ#1840962 のバグを参照してください。

前提条件

- Pod が追加ネットワークと同じ namespace にあること。

- OpenShift CLI (**oc**) のインストール。
- クラスターにログインする必要があります。
- SR-IOV Operator がインストールされ、SriovNetwork CR が定義されている必要があります。

手順

1. アノテーションを Pod オブジェクトに追加します。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。

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

   1 複数の追加ネットワークを指定するには、各ネットワークをカンマで区切ります。カ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ースをそのネットワークに割り当てます。

   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。

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

   1 NetworkAttachmentDefinition CR 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 CR 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 を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

   ```
   $ oc create -f <name>.yaml
   ```

3. オプション: アノテーションが Pod CR 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、以下のコマンドを入力します。

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

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

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

1 **k8s.v1.cni.cncf.io/networks-status** 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について説明します。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。

8.6.2. Non-Uniform Memory Access (NUMA) で配置された SR-IOV Pod の作成

NUMA で配置された SR-IOV Pod は、**restricted** または **single-numa-node** Topology Manager ポリシーで同じ NUMA ノードから割り当てられる SR-IOV および CPU リソースを制限することによって作成できます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- LatencySensitive プロファイルを有効にし、CPU マネージャーのポリシーを **static** に設定する。

手順

1. 以下の SR-IOV Pod 仕様を作成してから、YAML を **<name>-sriov-pod.yaml** ファイルに保存します。**<name>** をこの Pod の名前に置き換えます。

以下の例は、SR-IOV Pod 仕様を示しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> 1
spec:
  containers:
  - name: sample-container
    image: <image> 2
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" 3
        cpu: "2" 4
      requests:
        memory: "1Gi"
        cpu: "2"
```

1. **<name>** を、SR-IOV ネットワーク割り当て定義 CR の名前に置き換えます。
2. **<image>** を **sample-pod** イメージの名前に置き換えます。
3. Guaranteed QoS を指定して SR-IOV Pod を作成するには、**メモリー要求**に等しい**メモリー制限**を設定します。
4. Guaranteed QoS を指定して SR-IOV Pod を作成するには、**cpu 要求**に等しい **cpu 制限**を設定します。

2. 以下のコマンドを実行して SR-IOV Pod のサンプルを作成します。

```
$ oc create -f <filename> 1
```

1. **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. **sample-pod** が Guaranteed QoS を指定して設定されていることを確認します。

```
$ oc describe pod sample-pod
```

4. **sample-pod** が排他的 CPU を指定して割り当てられていることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. **sample-pod** に割り当てられる SR-IOV デバイスと CPU が同じ NUMA ノード上にあることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

8.7. 高パフォーマンスのマルチキャストの使用

Single Root I/O Virtualization (SR-IOV) ハードウェアネットワーク上でマルチキャストを使用できます。

8.7.1. 高パフォーマンスのマルチキャストの設定

OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーは、デフォルトネットワーク上の Pod 間のマルチキャストをサポートします。これは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のアプリケーションには適していません。インターネットプロトコルテレビ (IPTV) やマルチポイントビデオ会議など、ストリーミングメディアなどのアプリケーションでは、Single Root I/O Virtualization (SR-IOV) ハードウェアを使用してネイティブに近いパフォーマンスを提供できます。

マルチキャストに追加の SR-IOV インターフェースを使用する場合:

- マルチキャストパッケージは、追加の SR-IOV インターフェース経由で Pod によって送受信される必要があります。
- SR-IOV インターフェースに接続する物理ネットワークは、OpenShift Container Platform で制御されないマルチキャストルーティングとトポロジーを判別します。

8.7.2. マルチキャストでの SR-IOV インターフェースの使用

以下の手順では、サンプルのマルチキャスト用の SR-IOV インターフェースを作成します。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. SriovNetworkNodePolicy カスタムリソース (CR) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. SriovNetwork CR を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
```

```
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | 1
    {
      "type": "host-local", 2
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

1 2 DHCP を IPAM として設定する選択をした場合は、DHCP サーバー経由でデフォルトルート (**224.0.0.0/5** および **232.0.0.0/5**) をプロビジョニングするようにしてください。これにより、デフォルトのネットワークプロバイダーによって設定された静的なマルチキャストルートが上書きされます。

3. マルチキャストアプリケーションで Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] 1
    command: [ "sleep", "infinity"]
```

1 **NET_ADMIN** 機能は、アプリケーションがマルチキャスト IP アドレスを SR-IOV インターフェースに割り当てる必要がある場合にのみ必要です。それ以外の場合は省略できます。

8.8. DPDK および RDMA モードでの仮想機能 (VF) の使用

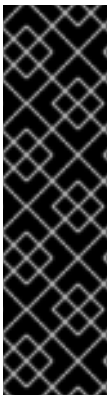
Single Root I/O Virtualization (SR-IOV) ネットワークハードウェアは、Data Plane Development Kit (DPDK) および Remote Direct Memory Access (RDMA) で利用できます。

8.8.1. DPDK および RDMA モードでの仮想機能 (VF) の使用例

重要

Data Plane Development Kit (DPDK) はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。



重要

Remote Direct Memory Access (RDMA) はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。

8.8.2. 前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてのログイン。
- SR-IOV ネットワーク Operator がインストールされていること。

8.8.3. Intel NIC を使用した DPDK モードでの仮想機能 (VF) の使用例

手順

1. 以下の SriovNetworkNodePolicy CR を作成してから、YAML を **intel-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
```

```
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci (1)
```

1 仮想機能 (VF) のドライバータイプを **vfio-pci** に指定します。

注記

SriovNetworkNodePolicy の各オプションに関する詳細は、「**SR-IOV ネットワークデバイスの設定**」セクションを参照してください。

+ SriovNetworkNodePolicy CR で指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

+ 設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

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

3. 以下の SriovNetwork CR を作成してから、YAML を **intel-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: "{}" (1)
  vlan: <vlan>
  resourceName: intelnics
```

1 IPAM CNI プラグインの空のオブジェクト **"{}"** を指定します。DPDK はユーザー空間モードで機能し、IP アドレスは必要ありません。

注記

SriovNetwork の各オプションに関する詳細は、「**SR-IOV の追加ネットワークの設定**」セクションを参照してください。

4. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

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

5. 以下の Pod 仕様を作成してから、YAML を **intel-dpdk-pod.yaml** ファイルに保存します。

-

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] 3
    volumeMounts:
    - mountPath: /dev/hugepages 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. SriovNetwork CR **intel-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様と SriovNetowrk CR の両方で変更します。

2. アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3. コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能を指定します。

4. hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS

7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。たとえば、カーネル引数 **default_hugepagesz=1GB**、**hugepagesz=1G** および **hugepages=16** を追加すると、**16*1Gi** hugepage がシステムの起動時に割り当てられます。

6. 以下のコマンドを実行して DPDK Pod を作成します。

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

8.8.4. Mellanox NIC を使用した DPDK モードでの仮想機能 (VF) の使用例

手順

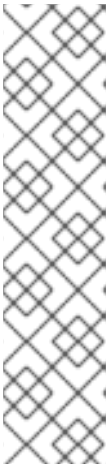
1. 以下の SriovNetworkNodePolicy CR を作成してから、YAML を **mlx-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

1 SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。Mellanox カードに許可される値は **1015**、**1017** です。

2 仮想機能 (VF) のドライバータイプを **netdevice** に指定します。Mellanox SR-IOV VF は、**vfio-pci** デバイスタイプを使用せずに DPDK モードで機能します。VF デバイスは、コンテナー内のカーネルネットワークインターフェースとして表示されます。

3 RDMA モードを有効にします。これは、DPDK モードで機能するために Mellanox カードで必要とされます。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、「**SR-IOV ネットワークデバイスの設定**」セクションを参照してください。

+ SriovNetworkNodePolicy CR で指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

+ 設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

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

3. 以下の SriovNetwork CR を作成してから、YAML を **mlx-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- (1)
    ...
  vlan: <vlan>
  resourceName: mlxnics
```

(1) IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

注記

SriovNetwork の各オプションに関する詳細は、「**SR-IOV の追加ネットワークの設定**」セクションを参照してください。

4. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

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

5. 以下の SR-IOV Pod 仕様を作成してから、YAML を **mlx-dpdk-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> (1)
```

```
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] 3
    volumeMounts:
    - mountPath: /dev/hugepages 4
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 SriovNetwork CR **mlx-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様および SriovNetowrk CR の両方で変更します。

2 アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3 コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能を指定します。

4 hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5 オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り

6. 以下のコマンドを実行して DPDK Pod を作成します。

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

8.8.5. Mellanox NIC を使った RDMA モードでの仮想機能 (VF) の例

RoCE (RDMA over Converged Ethernet) は、OpenShift Container Platform で RDMA を使用する場合に唯一サポートされているモードです。

手順

1. 以下の SriovNetworkNodePolicy CR を作成してから、YAML を **mlx-rdma-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。Mellanox カードに許可される値は **1015**、**1017** です。

(2) 仮想機能 (VF) のドライバータイプを **netdevice** に指定します。

(3) RDMA モードを有効にします。

注記

SriovNetworkNodePolicy の各オプションに関する詳細は、「**SR-IOV ネットワークデバイスの設定**」セクションを参照してください。

+ SriovNetworkNodePolicy CR で指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

+ 設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

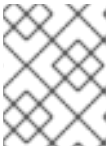
2. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3. 以下の SriovNetwork CR を作成してから、YAML を **mlx-rdma-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- (1)
    ...
  vlan: <vlan>
  resourceName: mlxnics
```

(1) IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。



注記

SriovNetwork の各オプションに関する詳細は、「**SR-IOV の追加ネットワークの設定**」セクションを参照してください。

4. 以下のコマンドを実行して SriovNetworkNodePolicy CR を作成します。

```
$ oc create -f mlx-rdma-network.yaml
```

5. 以下の SR-IOV Pod 仕様を作成してから、YAML を **mlx-rdma-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> (2)
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] (3)
    volumeMounts:
    - mountPath: /dev/hugepages (4)
      name: hugepage
    resources:
```

```
    limits:
      memory: "1Gi"
      cpu: "4" (5)
      hugepages-1Gi: "4Gi" (6)
    requests:
      memory: "1Gi"
      cpu: "4"
      hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

1. SriovNetwork CR **mlx-rdma-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様および SriovNetowrk CR の両方で変更します。

2. アプリケーションとアプリケーションが使用する RDMA ライブラリーが含まれる RDMA イメージを指定します。

3. コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能を指定します。

4. hugepage ボリュームを、**/dev/hugepages** の下にある RDMA Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. CPU の数を指定します。RDMA Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

6. hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、RDMA Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して RDMA Pod を作成します。

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

第9章 OPENSHIFT SDN デフォルト CNI ネットワークプロバイダー

9.1. OPENSHIFT SDN デフォルト CNI ネットワークプロバイダーについて

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。OpenShift SDN により、このような Pod ネットワークが確立され、メンテナンスされます。OpenShift SDN は Open vSwitch (OVS) を使用してオーバーレイネットワークを設定します。

OpenShift SDN では以下のように、Pod ネットワークを設定するための SDN モードを 3 つ提供します。

- **ネットワークポリシー**モードは、プロジェクト管理者が NetworkPolicy オブジェクトを使用して独自の分離ポリシーを設定することを可能にします。NetworkPolicy は OpenShift Container Platform 4.3 のデフォルトモードです。
- **マルチテナント** モードは、Pod およびサービスのプロジェクトレベルの分離を可能にします。異なるプロジェクトの Pod は、異なるプロジェクトの Pod およびサービスにパケットを送信したり、それらからパケットを受信したりすることができません。プロジェクトの分離を無効にし、クラスター全体のすべての Pod およびサービスにネットワークトラフィックを送信したり、それらの Pod およびサービスからネットワークトラフィックを受信したりすることができます。
- **サブネット** モードは、すべての Pod が他のすべての Pod およびサービスと通信できる Pod ネットワークを提供します。ネットワークポリシーモードは、サブネットモードと同じ機能を提供します。

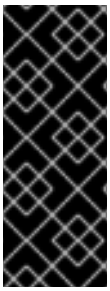
9.2. プロジェクトの EGRESS IP の設定

クラスター管理者は、OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーが 1 つ以上の egress IP アドレスをプロジェクトに割り当てるように設定できます。

9.2.1. プロジェクトの egress トラフィックについての egress IP アドレスの割り当て

プロジェクトの egress IP アドレスを設定することにより、指定されたプロジェクトからのすべての外部送信接続が同じ固定ソース IP アドレスを共有します。外部リソースは、egress IP アドレスに基づいて特定のプロジェクトからのトラフィックを認識できます。プロジェクトに割り当てられる egress IP アドレスは、トラフィックを特定の宛先に送信するために使用される egress ルーターとは異なります。

egress IP アドレスは、ノードのプライマリーネットワークインターフェースの追加 IP アドレスとして実装され、ノードのプライマリー IP アドレスと同じサブネットにある必要があります。



重要

egress IP アドレスは、**ifcfg-eth0** などのように Linux ネットワーク設定ファイルで設定することはできません。

一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェースで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

egress IP アドレスは、**NetNamespace** リソースの **egressIPs** パラメーターを設定して namespace に割り当てることができます。egress IP がプロジェクトに関連付けられた後に、 OpenShift SDN は 2 つの方法で Egress IP をホストに割り当てることを可能にします。

- **自動的に割り当てる** 方法では、egress IP アドレス範囲はノードに割り当てられます。
- **手動で割り当てる** 方法では、1 つ以上の egress IP アドレスの一覧がノードに割り当てられます。

egress IP アドレスを要求する namespace は、それらの egress IP アドレスをホストできるノードに一致し、egress IP アドレスはそれらのノードに割り当てられます。**egressIPs** パレメーターが **NetNamespace** リソースに設定されるものの、ノードがその egress IP アドレスをホストしない場合、namespace からの egress トラフィックはドロップされます。

ノードの高可用性は自動的に実行されます。egress IP アドレスをホストするノードが到達不可能であり、egress IP アドレスをホストできるノードがある場合、egress IP アドレスは新規ノードに移行します。到達不可能なノードが再びオンラインに戻ると、ノード間で egress IP アドレスのバランスを図るために egress IP アドレスは自動的に移行します。

重要

手動で割り当てられた egress IP アドレスと、自動的に割り当てられた egress IP アドレスは同じノードで使用することができません。IP アドレス範囲から egress IP アドレスを手動で割り当てる場合、その範囲を自動の IP 割り当てで利用可能にすることはできません。

9.2.1.1. 自動的に割り当てられた egress IP アドレスを使用する場合の考慮事項

egress IP アドレスの自動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressCIDRs** パラメーターを設定して、ノードでホストできる egress IP アドレスの範囲を指定します。OpenShift Container Platform は、指定する IP アドレス範囲に基づいて **HostSubnet** リソースの **egressIPs** パラメーターを設定します。
- 自動割り当てモードを使用する場合、namespace ごとに単一の egress IP アドレスのみがサポートされます。

namespace の egress IP アドレスをホストするノードに到達できない場合、OpenShift Container Platform は互換性のある egress IP アドレス範囲を持つ別のノードに egress IP アドレスを再割り当てします。自動割り当て方法は、追加の IP アドレスをノードに関連付ける柔軟性のある環境にインストールされたクラスターに最も適しています。

9.2.1.2. 手動で割り当てられた egress IP アドレスを使用する場合の考慮事項

egress IP アドレスに手動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressIPs** パラメーターを設定して、ノードでホストできる IP アドレスを指定します。
- namespace ごとに複数の egress IP アドレスがサポートされます。

namespace に複数の egress IP アドレスがある場合、最初の egress IP アドレスをホストするノードに到達できない場合、OpenShift Container Platform は最初の egress IP アドレスが再び到達可能になるまで、次に利用可能な egress IP アドレスの使用に自動的に切り替えます。

この方法は、パブリッククラウド環境にインストールされたクラスター用に推奨されます。この場合、追加の IP アドレスをノードに関連付ける上で制限がある場合があります。

9.2.2. namespace の自動的に割り当てられた egress IP アドレスの有効化

OpenShift Container Platform では、1 つ以上のノードで特定の namespace の egress IP アドレスの自動的な割り当てを有効にできます。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。

手順

1. 以下の JSON を使用して、**NetNamespace** リソースを egress IP アドレスで更新します。

```
$ oc patch netnamespace <project_name> --type=merge -p \ 1
 '{
   "egressIPs": [
     "<ip_address>" 2
   ]
 }'
```

1 プロジェクトのターゲットを指定します。

2 単一 egress IP アドレスを指定します。複数の IP アドレスの使用はサポートされません。

たとえば、**project1** を IP アドレスの 192.168.1.100 に、 **project2** を IP アドレスの 192.168.1.101 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100"]}'
$ oc patch netnamespace project2 --type=merge -p \
  '{"egressIPs": ["192.168.1.101"]}'
```

2. 以下の JSON を使用して、各ホストの **egressCIDRs** パラメーターを設定して egress IP アドレスをホストできるノードを示します。

```
$ oc patch hostsubnet <node_name> --type=merge -p \ 1
  '{
    "egressCIDRs": [
      "<ip_address_range_1>", "<ip_address_range_2>" 2
    ]
  }'
```

1 ノード名を指定します。

2 CIDR 形式で 1 つ以上の IP アドレス範囲を指定します。

たとえば、**node1** および **node2** を、192.168.1.0 から 192.168.1.255 の範囲で egress IP アドレスをホストするように設定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
```

OpenShift Container Platform はバランスを取りながら特定の egress IP アドレスを利用可能なノードに自動的に割り当てます。この場合、egress IP アドレス 192.168.1.100 を **node1** に、egress IP アドレス 192.168.1.101 を **node2** に割り当て、その逆も行います。

9.2.3. namespace の手動で割り当てられた egress IP アドレスの設定

OpenShift Container Platform で、1 つ以上の egress IP アドレスを namespace に関連付けることができます。

前提条件

- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。

手順

1. 以下の JSON オブジェクトを必要な IP アドレスで指定して、**NetNamespace** リソースを更新します。

```
$ oc patch netnamespace <project> --type=merge -p \ 1
  '{
    "egressIPs": [ 2
      "<ip_address>"
    ]
  }'
```

1 プロジェクトのターゲットを指定します。

2 1 つ以上の egress IP アドレスを指定します。**egressIPs** パラメーターは配列です。

たとえば、**project1** プロジェクトを **192.168.1.100** の IP アドレスに割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100"]}'
```

egressIPs を異なるノードの 2 つ以上の IP アドレスに設定し、高可用性を確保することができます。複数の egress IP アドレスが設定される場合、Pod は egress の一覧にある最初の IP を使用しますが、IP アドレスをホストするノードが失敗する場合、Pod は短時間の遅延の後に一覧にある次の IP の使用に切り替えます。

2. egress IP をノードホストに手動で割り当てます。**egressIPs** パラメーターを、ノードホストの **HostSubnet** オブジェクトに設定します。以下の JSON を使用して、そのノードホストに割り当てる必要のある任意の数の IP を含めることができます。

```
$ oc patch hostsubnet <node_name> --type=merge -p \ 1
  '{
```

```
  "egressIPs": [ 2
    "<ip_address_1>",
    "<ip_address_N>"
    ]
}'
```

1 ノードの名前を指定します。

2 1 つ以上の egress IP アドレスを指定します。**egressIPs** フィールドは配列です。

たとえば、**node1** に Egress IP **192.168.1.100**、 **192.168.1.101**、および **192.168.1.102** が設定されるように指定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

直前の例では、**project1** のすべての egress トラフィックは、指定された egress IP をホストするノードにルーティングされてから、その IP アドレスに (NAT を使用して) 接続されます。

9.3. 外部 IP アドレスへのアクセスを制御するための EGRESS ファイアウォールの設定

クラスター管理者は、OpenShift Container Platform クラスター外に出るプロジェクトのプロジェクについて、egress トラフィックを制限する egress ファイアウォールを作成できます。

9.3.1. egress ファイアウォールのプロジェクトでの機能

クラスター管理者は、 **egress ファイアウォール** を使用して、一部またはすべての Pod がクラスター内からアクセスできる外部ホストを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Container Platform クラスター外にある内部ホストへの接続を開始できないようにする。
- Pod は OpenShift Container Platform クラスター外の指定された内部サブネットまたはホストにアクセスできません。
- Pod は特定の外部ホストにのみ接続することができます。

egress ファイアウォールポリシーは、EgressNetworkPolicy カスタムリソース (CR) オブジェクトを作成し、IP アドレス範囲を CIDR 形式で指定するか、または DNS 名を指定して設定します。たとえば、指定された IP 範囲へのあるプロジェクトへのアクセスを許可する一方で、別のプロジェクトへの同じアクセスを拒否することができます。または、アプリケーション開発者の (Python) pip mirror からの更新を制限したり、更新を承認されたソースからの更新のみに強制的に制限したりすることができます。

重要

egress ファイアウォールポリシーを設定するには、ネットワークポリシーまたはマルチテナントモードのいずれかを使用するように OpenShift SDN を設定する必要があります。

ネットワークポリシーモードを使用している場合、egress ポリシーは namespace ごとに 1 つのポリシーとのみ互換性を持ち、グローバルプロジェクトなどのネットワークを共有するプロジェクトでは機能しません。

警告

egress ファイアウォールルールは、ルーターを通過するトラフィックには適用されません。ルート CR オブジェクトを作成するパーミッションを持つユーザーは、禁止されている宛先を参照するルートを作成することにより、egress ネットワークポリシールールをバイパスできます。

9.3.1.1. egress ファイアウォールの制限

egress ファイアウォールには以下の制限があります。

- いずれのプロジェクトも複数の EgressNetworkPolicy オブジェクトを持つことができません。
- 最大 50 のルールを持つ最大 1 EgressNetworkPolicy オブジェクトはプロジェクトごとに定義できます。
- **default** プロジェクトは egress ネットワークポリシーを使用できません。
- マルチテナントモードで OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用する場合、以下の制限が適用されます。
  - グローバルプロジェクトは egress ファイアウォールを使用できません。**oc adm pod-network make-projects-global** コマンドを使用して、プロジェクトをグローバルにすることができます。
  - **oc adm pod-network join-projects** コマンドを使用してマージされるプロジェクトでは、結合されたプロジェクトのいずれでも egress ファイアウォールを使用することはできません。

上記の制限のいずれかに違反すると、プロジェクトの egress ネットワークポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

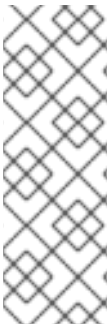
9.3.1.2. egress ネットワークポリシールールのマッチング順序

egress ネットワークポリシールールは、最初から最後へと定義された順序で評価されます。Pod からの egress 接続に一致する最初のルールが適用されます。この接続では、後続のルールは無視されます。

9.3.1.3. DNS (Domain Name Server) 解決の仕組み

egress ファイアウォールポリシールールのいずれかで DNS 名を使用する場合、ドメイン名の適切な解決には、以下の制限が適用されます。

- ドメイン名の更新は、ローカルの非権威サーバーのドメインの TTL (time to live) 値に基づいてポーリングされます。
- Pod は、必要に応じて同じローカルネームサーバーからドメインを解決する必要があります。そうしない場合、egress ファイアウォールコントローラーと Pod によって認識されるドメインの IP アドレスが異なる可能性があります。ホスト名の IP アドレスが異なる場合、egress ファイアウォールは一貫して実行されないことがあります。
- egress ファイアウォールコントローラーおよび Pod は同じローカルネームサーバーを非同期にポーリングするため、Pod は egress コントローラーが実行する前に更新された IP アドレスを取得する可能性があります。これにより、競合状態が生じます。この現時点の制限により、EgressNetworkPolicy オブジェクトのドメイン名の使用は、IP アドレスの変更が頻繁に生じないドメインの場合にのみ推奨されます。



注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェースに Pod が常にアクセスできるようにします。

ドメイン名を egress ファイアウォールで使用し、DNS 解決がローカルノード上の DNS サーバーによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。

9.3.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト

以下の YAML は EgressNetworkPolicy CR オブジェクトについて説明しています。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1 egress ファイアウォールポリシーの **name** を指定します。

2 以下のセクションで説明されているように、egress ネットワークポリシールールのコレクションを指定します。

9.3.2.1. EgressNetworkPolicy ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** キーは、単一または複数のオブジェクトの配列を予想します。

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns-name> 4
```

1. ルールのタイプを指定します。値には **Allow** または **Deny** のいずれかを指定する必要があります。
2. ルールの **cidrSelector** キーまたは **dnsName** キーのいずれかの値を指定します。ルールで両方のキーを使用することはできません。
3. CIDR 形式の IP アドレス範囲を指定します。
4. ドメイン名を指定します。

9.3.2.2. EgressNetworkPolicy CR オブジェクトの例

以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default-rules (1)
spec:
  egress: (2)
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1. ポリシーオブジェクトの名前。
2. egress ファイアウォールポリシールールオブジェクトのコレクション。

9.3.3. egress ファイアウォールポリシーオブジェクトの作成

クラスター管理者は、プロジェクトの egress ファイアウォールポリシーオブジェクトを作成できます。

重要

プロジェクトに EgressNetworkPolicy オブジェクトがすでに定義されている場合、既存のポリシーを編集して egress ファイアウォールルールを変更する必要があります。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) のインストール。
- クラスター管理者としてクラスターにログインする必要があります。

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。 この場合、**<policy-name>** は egress ポリシールールを記述します。

   b. 作成したファイルで、egress ポリシーオブジェクトを定義します。

2. 以下のコマンドを入力してポリシーオブジェクトを作成します。

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

   以下の例では、新規の EgressNetworkPolicy オブジェクトが **project1** という名前のプロジェクトに作成されます。

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

   出力例

   ```
   egressnetworkpolicy.network.openshift.io/default-rules created
   ```

3. オプション: 後に変更できるように **<policy-name>.yaml** を保存します。

9.4. プロジェクトの EGRESS ファイアウォールの編集

クラスター管理者は、既存の egress ファイアウォールのネットワークトラフィックルールを変更できます。

9.4.1. EgressNetworkPolicy オブジェクトの編集

クラスター管理者は、プロジェクトの egress ファイアウォールを更新できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- OpenShift CLI (**oc**) のインストール。
- クラスター管理者としてクラスターにログインする必要があります。

手順

プロジェクトの既存の egress ネットワークポリシーオブジェクトを編集するには、以下の手順を実行します。

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。**<project>** をプロジェクトの名前に置き換えます。

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

2. オプション: egress ネットワークファイアウォールの作成時に EgressNetworkPolicy オブジェクトのコピーを保存しなかった場合には、以下のコマンドを入力してコピーを作成します。

```
$ oc get -n <project> \ 1
  egressnetworkpolicy <name> \ 2
  -o yaml > <filename>.yaml 3
```

1. **<project>** をプロジェクトの名前に置き換えます。
2. **<name>** をオブジェクトの名前に置き換えます。
3. **<filename>** をファイルの名前に置き換え、YAML を保存します。

3. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを置き換えます。**<filename>** を、更新された EgressNetworkPolicy オブジェクトを含むファイルの名前に置き換えます。

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

9.4.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト

以下の YAML は EgressNetworkPolicy CR オブジェクトについて説明しています。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> 1
spec:
  egress: 2
    ...
```

1. egress ファイアウォールポリシーの **name** を指定します。
2. 以下のセクションで説明されているように、egress ネットワークポリシールールのコレクションを指定します。

9.4.2.1. EgressNetworkPolicy ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** キーは、単一または複数のオブジェクトの配列を予想します。

```
egress:
- type: <type> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns-name> 4
```

1. ルールのタイプを指定します。値には **Allow** または **Deny** のいずれかを指定する必要があります。
2. ルールの **cidrSelector** キーまたは **dnsName** キーのいずれかの値を指定します。ルールで両方のキーを使用することはできません。
3. CIDR 形式の IP アドレス範囲を指定します。

4 ドメイン名を指定します。

9.4.2.2. EgressNetworkPolicy CR オブジェクトの例

以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default-rules 1
spec:
  egress: 2
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 ポリシーオブジェクトの名前。

2 egress ファイアウォールポリシールールオブジェクトのコレクション。

9.5. プロジェクトからの EGRESS ファイアウォールの削除

クラスター管理者は、プロジェクトから egress ファイアウォールを削除して、OpenShift Container Platform クラスター外に出るプロジェクトからネットワークトラフィックについてのすべての制限を削除できます。

9.5.1. EgressNetworkPolicy オブジェクトの削除

クラスター管理者は、プロジェクトから Egress ファイアウォールを削除できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- OpenShift CLI (**oc**) のインストール。
- クラスター管理者としてクラスターにログインする必要があります。

手順

プロジェクトの egress ネットワークポリシーオブジェクトを削除するには、以下の手順を実行します。

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。**<project>** をプロジェクトの名前に置き換えます。

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

2. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを削除します。**<project>** をプロジェクトの名前に、 **<name>** をオブジェクトの名前に置き換えます。

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

9.6. マルチキャストの使用

9.6.1. マルチキャストについて

IP マルチキャストを使用すると、データは多数の IP アドレスに同時に配信されます。

重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform Pod 間のマルチキャストトラフィックはデフォルトで無効にされています。OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用している場合、プロジェクトごとにマルチキャストを有効にできます。

networkpolicy 分離モードで OpenShift SDN ネットワークプラグインを使用する場合:

- Pod によって送信されるマルチキャストパケットは、NetworkPolicy ポリシーに関係なく、プロジェクトの他のすべての Pod に送信されます。Pod はユニキャストで通信できない場合でもマルチキャストで通信できます。
- 1 つのプロジェクトの Pod によって送信されるマルチキャストパケットは、NetworkPolicy オブジェクトがプロジェクト間の通信を許可する場合であっても、それ以外のプロジェクトの Pod に送信されることはありません。

multinenant 分離モードで OpenShift SDN ネットワークプラグインを使用する場合:

- Pod で送信されるマルチキャストパケットはプロジェクト内の他のすべての Pod に送信されます。
- あるプロジェクトの Pod によって送信されるマルチキャストパケットは、各プロジェクトが結合し、マルチキャストが結合した各プロジェクトで有効にされている場合にのみ、他のプロジェクトの Pod に送信されます。

9.6.2. Pod 間のマルチキャストの有効化

プロジェクトの Pod でマルチキャストを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行し、プロジェクトのマルチキャストを有効にします。

```
$ oc annotate netnamespace <namespace> \ 1
    netnamespace.network.openshift.io/multicast-enabled=true
```

1 マルチキャストを有効にする必要のあるプロジェクトの **namespace**。

9.6.3. Pod 間のマルチキャストの無効化

プロジェクトの Pod でマルチキャストを無効にすることができます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行して、マルチキャストを無効にします。

```
$ oc annotate netnamespace <namespace> \ 1
    netnamespace.network.openshift.io/multicast-enabled-
```

1 マルチキャストを無効にする必要のあるプロジェクトの **namespace**。

9.7. OPENSHIFT SDN を使用したネットワーク分離の設定

クラスターが OpenShift SDN CNI プラグインのマルチテナント分離モードを使用するように設定されている場合、各プロジェクトはデフォルトで分離されます。ネットワークトラフィックは、マルチテナント分離モードでは、異なるプロジェクトの Pod およびサービス間で許可されません。

プロジェクトのマルチテナント分離の動作を 2 つの方法で変更することができます。

- 1 つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。
- プロジェクトのネットワーク分離を無効にできます。これはグローバルにアクセスできるようになり、他のすべてのプロジェクトの Pod およびサービスからのネットワークトラフィックを受け入れます。グローバルにアクセス可能なプロジェクトは、他のすべてのプロジェクトの Pod およびサービスにアクセスできます。

9.7.1. 前提条件

- クラスターは、マルチテナント分離ノードで OpenShift SDN Container Network Interface (CNI) プラグインを使用するように設定されている必要があります。

9.7.2. プロジェクトの結合

2 つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. 以下のコマンドを使用して、プロジェクトを既存のプロジェクトネットワークに参加させます。

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

   または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

2. オプション: 以下のコマンドを実行し、結合した Pod ネットワークを表示します。

```
$ oc get netnamespaces
```

   同じ Pod ネットワークのプロジェクトには、**NETID** 列に同じネットワーク ID があります。

9.7.3. プロジェクトの分離

他のプロジェクトの Pod およびサービスがその Pod およびサービスにアクセスできないようにするためにプロジェクトを分離することができます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- クラスターのプロジェクトを分離するには、以下のコマンドを実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

  または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

9.7.4. プロジェクトのネットワーク分離の無効化

プロジェクトのネットワーク分離を無効にできます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- プロジェクトの以下のコマンドを実行します。

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

9.8. KUBE-PROXY の設定

Kubernetes メットワークプロキシー (kube-proxy) は各ノードで実行され、Cluster Network Operator (CNO) で管理されます。kube-proxy は、サービスに関連付けられたエンドポイントの接続を転送するためのネットワークルールを維持します。

9.8.1. iptables ルールの同期について

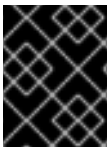
同期の期間は、Kubernetes ネットワークプロキシー (kube-proxy) がノードで iptables ルールを同期する頻度を定めます。

同期は、以下のイベントのいずれかが生じる場合に開始します。

- サービスまたはエンドポイントのクラスターへの追加、またはクラスターからの削除などのイベントが発生する。
- 最後の同期以後の時間が kube-proxy に定義される同期期間を超過している。

9.8.2. kube-proxy 設定パラメーター

以下の **kubeProxyConfig** パラメーターを変更することができます。



重要

OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、**iptablesSyncPeriod** パラメーターを調整する必要はなくなりました。

表9.1 パラメーター

パラメーター	説明	値	デフォルト
iptablesSyncPeriod	**iptables** ルールの更新期間。	**30s** または **2m** などの期間。有効なサフィックスには、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time ドキュメントで説明されています。	**30s**
proxyArguments.iptables-min-sync-period	**iptables** ルールを更新する前の最小期間。このパラメーターにより、更新の頻度が高くなり過ぎないようにできます。デフォルトでは、**iptables** ルールに影響する変更が生じるとすぐに、更新が開始されます。	**30s** または **2m** などの期間。有効なサフィックスには、**s**、**m**、および **h** が含まれ、これらについては、Go Package time で説明されています。	**0s**

9.8.3. kube-proxy 設定の変化

クラスターの Kubernetes ネットワークプロキシー設定を変更することができます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールで実行中のクラスターにログインします。

手順

1. 以下のコマンドを実行して、**Network.operator.openshift.io** カスタムリソース (CR) を編集します。

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

2. 以下のサンプル CR のように、kube-proxy 設定への変更内容で、CR の**kubeProxyConfig** パラメーターを変更します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period: ["30s"]
```

3. ファイルを保存し、テキストエディターを編集します。
構文は、ファイルを保存し、エディターを終了する際に **oc** コマンドによって検証されます。
変更内容に構文エラーが含まれる場合、エディターはファイルを開き、エラーメッセージを表示します。

4. 以下のコマンドを実行して、設定の更新を確認します。

```
$ oc get networks.operator.openshift.io -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
```

```
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
        - 30s
    serviceNetwork:
    - 172.30.0.0/16
  status: {}
kind: List
```

5. オプション: 以下のコマンドを実行し、Cluster Network Operator が設定変更を受け入れていることを確認します。

```
$ oc get clusteroperator network
```

出力例

```
NAME      VERSION     AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.1.0-0.9   True        False         False      1m
```

設定の更新が正常に適用されると、**AVAILABLE** フィールドが **True** になります。

第10章 ルートの作成

10.1. ルート設定

10.1.1. ルートのタイムアウトの設定

Service Level Availability (SLA) で必要とされる、低タイムアウトが必要なサービスや、バックエンドでの処理速度が遅いケースで高タイムアウトが必要なサービスがある場合は、既存のルートに対してデフォルトのタイムアウトを設定することができます。

前提条件

- 実行中のクラスターでデプロイ済みの Ingress コントローラーが必要になります。

手順

1. **oc annotate** コマンドを使用して、ルートにタイムアウトを追加します。

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1 サポートされる時間単位は、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、時間 (h)、または日 (d) です。

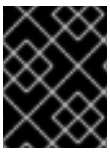
以下の例では、2 秒のタイムアウトを **myroute** という名前のルートに設定します。

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

10.1.2. HTTP Strict Transport Security の有効化

HTTP Strict Transport Security (HSTS) ポリシーは、ホストで HTTPS トラフィックのみを許可するセキュリティーの拡張機能です。デフォルトで、すべての HTTP 要求はドロップされます。これは、web サイトとの対話の安全性を確保したり、ユーザーのためにセキュアなアプリケーションを提供するのに役立ちます。

HSTS が有効にされると、HSTS はサイトから Strict Transport Security ヘッダーを HTTPS 応答に追加します。リダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用し、HTTP を HTTPS に送信するようにします。ただし、HSTS が有効にされている場合は、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 に変更するためにリダイレクトの必要がなくなります。これはクライアントでサポートされる必要はなく、**max-age=0** を設定することで無効にできます。



重要

HSTS はセキュアなルート (edge termination 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。

手順

- ルートに対して HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge termination または re-encrypt ルートに追加します。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
```
(1) (2) (3)

(1) **max-age** は唯一の必須パラメーターです。これは、HSTS ポリシーが有効な期間 (秒単位) を測定します。クライアントは、ホストから HSTS ヘッダーのある応答を受信する際には常に **max-age** を更新します。**max-age** がタイムアウトになると、クライアントはポリシーを破棄します。

(2) **includeSubDomains** はオプションです。これが含まれる場合、クライアントに対し、ホストのすべてのサブドメインがホストと同様に処理されるように指示します。

(3) **preload** はオプションです。**max-age** が 0 より大きい場合、**preload** を **haproxy.router.openshift.io/hsts_header** に組み込むことにより、外部サービスはこのサイトをそれぞれの HSTS プリロード一覧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一覧を使用し、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** 設定がない場合、ブラウザーはヘッダーを取得するために HTTPS 経由でサイトと通信している必要があります。

10.1.3. スループット関連の問題のトラブルシューティング

OpenShift Container Platform でデプロイされるアプリケーションでは、特定のサービス間で非常に長い待ち時間が発生するなど、ネットワークのスループットの問題が生じることがあります。

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 Pod 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。待ち時間は、ノードのインターフェースが他の Pod や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オーバーロードする場合に OpenShift Container Platform で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 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。理想的には、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。以下のように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 (式から SDN を排除する)。

```
$ 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。ボトルネックの特定を試行するには、最初に Pod から、次にノードからツールを実行します。
  - iperf のインストールおよび使用についての詳細は、こちらの Red Hat ソリューション を参照してください。

10.1.4. Cookie に使用によるルートのステートフル性の維持

OpenShift Container Platform 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。

OpenShift Container Platform は Cookie を使用してセッションの永続化を設定できます。Ingress コントローラー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 コントローラーに対し、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。

10.1.4.1. 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。そのためサーバーがオーバーロードしている場合には、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. 必要な 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_annotation>"
```

   たとえば、**my_cookie_annotation** というアノテーションで **my_route** に **my_cookie** という Cookie 名の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/my_cookie="-my_cookie_annotation"
```

2. Cookie を保存し、ルートにアクセスします。

```
$ curl $my_route -k -c /tmp/my_cookie
```

10.1.5. ルート固有のアノテーション

Ingress コントローラーは、公開するすべてのルートのデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。

表10.1 ルートアノテーション

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/balance	ロードバランシングアルゴリズムを設定します。使用できるオプションは **source**、**roundrobin**、および **leastconn** です。	パススルールートの **ROUTER_TCP_BALANCE_SCHEME** です。それ以外の場合は **ROUTER_LOAD_BALANCE_ALGORITHM** を使用します。
haproxy.router.openshift.io/disable_cookies	関連の接続を追跡する cookie の使用を無効にします。**true** または **TRUE** に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
router.openshift.io/cookie_name	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	
haproxy.router.openshift.io/pod-concurrent-connections	ルーターからバッキングされる Pod に対して許容される接続最大数を設定します。注意: Pod が複数ある場合には、それぞれに対応する接続数を設定できますが、ルーターが複数ある場合には、ルーター間の連携がなく、それぞれの接続回数はルーターの数と同じとなります。ただし、複数のルーターがある場合は、それらのルーター間で調整は行われず、それぞれがこれに複数回接続する可能性があります。設定されていない場合または 0 に設定されている場合には制限はありません。	
haproxy.router.openshift.io/rate-limit-connections	レート制限機能を有効にするために **true** または **TRUE** を設定します。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	IP アドレスで共有される同時 TCP 接続の数を制限します。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	IP アドレスが HTTP 要求を実行できるレートを制限します。	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	IP アドレスが TCP 接続を行うレートを制限します。	
haproxy.router.openshift.io/timeout	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
router.openshift.io/haproxy.health.check.interval	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	ルートのホワイトリストを設定します。	
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	

注記

環境変数を編集することはできません。

ルート設定のカスタムタイムアウト

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
...
```

(1) HAProxy 対応の単位 (**us**、**ms**、**s**、 **m**、**h**、**d**) で新規のタイムアウトを指定します。単位が指定されていない場合は、**ms** がデフォルトになります。

注記

passthrough ルートのサーバー側のタイムアウトを低く設定し過ぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

10.2. セキュリティー保護されたルート

以下のセクションでは、カスタム証明書を使用して re-encrypt および edge ルートを作成する方法を説明します。

重要

パブリックエンドポイントを使用して Microsoft Azure にルートを作成する場合、リソース名は制限されます。特定の用語を使用するリソースを作成することはできません。Azure が制限する語の一覧は、Azure ドキュメントの「Resolve reserved resource name errors」 を参照してください。

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

oc create route コマンドを使用し、カスタム証明書と共に reencrypt TLS termination を使用してセキュアなルートを設定できます。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。 ここで、証明書はルートホストに対して有効である必要があります。
- 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 Service リソースが必要です。

注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスフレーズを削除するには、以下のコマンドを使用します。

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

手順

この手順では、カスタム証明書および reencrypt TLS termination を使用して Route リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提としています。また、Ingress コントローラーがサービスの証明書を信頼できるように宛先 CA 証明書を指定する必要もあります。必要な場合には、証明書チェーンを完了するために CA 証明書を指定することもできます。**tls.crt**、 **tls.key**、**cacert.crt**、および (オプションで) **ca.crt** を実際のパス名に置き換えます。**frontend** を、公開する必要のある **Service** リソースに置き換えます。**www.example.com** を適切な名前に置き換えます。

- reencrypt TLS 終端およびカスタム証明書を使用してセキュアな Route リソースを作成します。

```
$ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
```

結果として生成される Route リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: frontend
```

```
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

他のオプションについては、**oc create route reencrypt --help** を参照してください。

10.2.2. カスタム証明書を使用した edge ルートの作成

oc create route コマンドを使用し、edge TLS termination とカスタム証明書を使用してセキュアなルートを設定できます。edge ルートの場合、Ingress コントローラーは、トラフィックを宛先 Pod に転送する前に TLS 暗号を終了します。ルートは、Ingress コントローラーがルートに使用する TLS 証明書およびキーを指定します。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。 ここで、証明書はルートホストに対して有効である必要があります。
- 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。
- 公開する必要のある Service リソースが必要です。

注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスフレーズを削除するには、以下のコマンドを使用します。

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

手順

この手順では、カスタム証明書および edge TLS termination を使用して Route リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提としています。必要な場合には、証明書チェーンを完了するために CA 証明書を指定する

こともできます。**tls.crt**、 **tls.key**、および (オプションで) **ca.crt** を実際のパス名に置き換えます。**frontend** を、公開する必要のある Service リソースの名前に置き換えます。**www.example.com** を適切な名前に置き換えます。

- edge TLS termination およびカスタム証明書を使用して、セキュアな Route リソースを作成します。

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

結果として生成される Route リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

他のオプションについては、**oc create route edge --help** を参照してください。

第11章 INGRESS クラスタートラフィックの設定

11.1. INGRESS クラスタートラフィックの設定の概要

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする以下の方法を提供します。

以下の方法が推奨されます。以下は、これらの方法の優先される順です。

- HTTP/HTTPS を使用する場合は Ingress コントローラーを使用する。
- HTTPS 以外の TLS で暗号化されたプロトコルを使用する場合、たとえば、SNI ヘッダーを使用する TLS の場合は、Ingress コントローラーを使用します。
- それ以外の場合は、ロードバランサー、外部 IP、または **NodePort**を使用します。

方法	目的
Ingress コントローラーの使用	HTTP/HTTPS トラフィックおよび HTTPS 以外の TLS で暗号化されたプロトコル (TLS と SNI ヘッダーの使用など) へのアクセスを許可します。
ロードバランサーサービスを使用した外部 IP の自動割り当て	プールから割り当てられた IP アドレスを使った非標準ポートへのトラフィックを許可します。
外部 IP のサービスへの手動割り当て	特定の IP アドレスを使った非標準ポートへのトラフィックを許可します。
NodePort の設定	クラスターのすべてのノードでサービスを公開します。

11.2. サービスの EXTERNALIP の設定

クラスター管理者は、トラフィックをクラスター内のサービスに送信できるクラスター外の IP アドレスブロックを指定できます。

この機能は通常、ベアメタルハードウェアにインストールされているクラスターに最も役立ちます。

11.2.1. 前提条件

- ネットワークインフラストラクチャーは、外部 IP アドレスのトラフィックをクラスターにルーティングする必要があります。

第12章 EXTERNALIP について

クラウド以外の環境では、OpenShift Container Platform は **ExternalIP** 機能を使用して外部 IP アドレスのサービス **spec.externalIPs** フィールドへの割り当てをサポートします。これにより、サービスに割り当てられた追加の仮想 IP アドレスが公開されます。これはクラスターに定義されたサービスネットワーク外にある可能性があります。**type=NodePort** が設定されたサービスと同様に外部 IP 機能で設定されたサービスにより、トラフィックを負荷分散のためにローカルノードに転送することができます。

ネットワークインフラストラクチャーを設定し、定義する外部 IP アドレスブロックがクラスターにルーティングされるようにする必要があります。

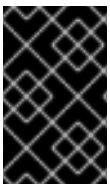
OpenShift Container Platform は以下の機能を追加して Kubernetes の ExternalIP 機能を拡張します。

- 設定可能なポリシーによる外部 IP アドレスの使用の制限
- 要求時の外部 IP アドレスのサービスへの自動割り当て

デフォルトでは、**cluster-admin** 権限を持つユーザーのみが、**spec.externalIPs[]** が外部 IP アドレスブロック内に定義された IP アドレスに設定されたサービスを作成できます。

警告

ExternalIP 機能の使用はデフォルトで無効にされます。これは、外部 IP アドレスへのクラスター内のトラフィックがそのサービスに転送されるため、セキュリティー上のリスクを生じさせる可能性があります。これにより、クラスターユーザーは外部リソースについての機密性の高いトラフィックをインターセプトできるようになります。



重要

この機能は、クラウド以外のデプロイメントでのみサポートされます。クラウドデプロイメントの場合、クラウドの自動デプロイメントのためにロードバランサーサービスを使用し、サービスのエンドポイントをターゲットに設定します。

以下の方法で外部 IP アドレスを割り当てることができます。

外部 IP の自動割り当て

OpenShift Container Platform は、**spec.type=LoadBalancer** を設定してサービスを作成する際に、IP アドレスを **autoAssignCIDRs** CIDR ブロックから **spec.externalIPs[]** 配列に自動的に割り当てます。この場合、OpenShift Container Platform はロードバランサーサービスタイプのクラウド以外のバージョンを実装し、IP アドレスをサービスに割り当てます。自動割り当てはデフォルトで無効にされており、以下のセクションで説明されているように、これはクラスター管理者が設定する必要があります。

外部 IP の手動割り当て

OpenShift Container Platform はサービスの作成時に **spec.externalIPs[]** 配列に割り当てられた IP アドレスを使用します。別のサービスによってすでに使用されている IP アドレスを指定することはできません。

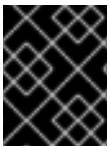
12.1. EXTERNALIP の設定

OpenShift Container Platform での外部 IP アドレスの使用は、**cluster** という名前の **Network.config.openshift.io** CR の以下のフィールドで管理されます。

- **spec.externalIP.autoAssignCIDRs** は、サービスの外部 IP アドレスを選択する際にロードバランサーによって使用される IP アドレスブロックを定義します。OpenShift Container Platform は、自動割り当て用の単一 IP アドレスブロックのみをサポートします。これは、ExternalIP をサービスに手動で割り当てる際に、制限された数の共有 IP アドレスのポート領域を管理しなくてはならない場合よりも単純になります。自動割り当てが有効な場合には、**spec.type=LoadBalancer** が設定されたサービスには外部 IP アドレスが割り当てられます。

- **spec.externalIP.policy** は、IP アドレスを手動で指定する際に許容される IP アドレスブロックを定義します。OpenShift Container Platform は、**spec.externalIP.autoAssignCIDRs** で定義される IP アドレスブロックにポリシールールを適用しません。

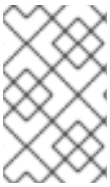
ルーティングが正しく行われると、設定された外部 IP アドレスブロックからの外部トラフィックは、サービスが公開する TCP ポートまたは UDP ポートを介してサービスのエンドポイントに到達できます。



重要

割り当てる IP アドレスブロックがクラスター内の 1 つ以上のノードで終了することを確認する必要があります。

OpenShift Container Platform は IP アドレスの自動および手動割り当ての両方をサポートしており、それぞれのアドレスは 1 つのサービスの最大数に割り当てられることが保証されます。これにより、各サービスは、ポートが他のサービスで公開されているかによらず、自らの選択したポートを公開できます。



注記

OpenShift Container Platform の **autoAssignCIDRs** で定義された IP アドレスブロックを使用するには、ホストのネットワークに必要な IP アドレスの割り当ておよびルーティングを設定する必要があります。

以下の YAML は、外部 IP が設定されたサービスについて説明しています。

spec.externalIPs[] が設定されたサービスオブジェクトの例

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
```

```
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

12.2. 外部 IP アドレスの割り当ての制限

クラスター管理者は、IP アドレスブロックを指定して許可および拒否できます。

spec.ExternalIP.policy フィールドを指定して、**policy** オブジェクトが定義された IP アドレスポリシーを設定します。ポリシーオブジェクトには以下の形があります。

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

ポリシーの制限を設定する際に、以下のルールが適用されます。

- **policy={}** が設定される場合、**spec.ExternalIPs[]** が設定されているサービスの作成は失敗します。これは OpenShift Container Platform のデフォルトです。
- **policy=null** が設定される場合、**spec.ExternalIPs[]** が IP アドレスに設定されるサービスの作成は許可されます。
- **policy** が設定され、**policy.allowedCIDRs[]** または **policy.rejectedCIDRs[]** のいずれかが設定される場合、以下のルールが適用されます。
  - **allowedCIDRs[]** と **rejectedCIDRs[]** の両方が設定される場合、**rejectedCIDRs[]** が **allowedCIDRs[]** よりも優先されます。
  - **allowedCIDRs[]** が設定される場合、 **spec.ExternalIPs[]** が設定されているサービスの作成は、指定された IP アドレスが許可される場合にのみ正常に実行されます。
  - **rejectedCIDRs[]** が設定される場合、 **spec.ExternalIPs[]** が設定されているサービスの作成は、指定された IP アドレスが拒否されていない場合にのみ正常に実行されます。

12.3. ポリシーオブジェクトの例

以下に続く例では、複数のポリシー設定の例を示します。

- 以下の例では、ポリシーは OpenShift Container Platform が外部 IP アドレスが指定されたサービスを作成するのを防ぎます。

  サービスの spec.externalIPs[] に指定された値を拒否するポリシーの例

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

```
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: {}
  ...
```

- 以下の例では、**allowedCIDRs** および **rejectedCIDRs** フィールドの両方が設定されます。

  許可される、および拒否される CIDR ブロックの両方を含むポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
      - 172.16.66.10/23
      rejectedCIDRs:
      - 172.16.66.10/24
  ...
```

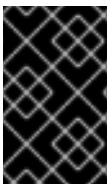
- 以下の例では、**policy** は **null** に設定されます。**null** に設定されている場合、**oc get networks.config.openshift.io -o yaml** を入力して設定オブジェクトを検査する際に、**policy** フィールドは出力に表示されません。

  サービスの spec.externalIPs[] に指定された値を許可するポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...
```

第13章 EXTERNALIP アドレスブロックの設定

ExternalIP アドレスブロックの設定は、**cluster** という名前の Network カスタムリソース (CR) で定義されます。ネットワーク CR は **config.openshift.io** API グループに含まれます。



重要

クラスターのインストール時に、Cluster Version Operator (CVO) は **cluster** という名前のネットワーク CR を自動的に作成します。このタイプのその他の CR オブジェクトの作成はサポートされていません。

以下の YAML は ExternalIP 設定について説明しています。

cluster という名前の network.config.openshift.io CR

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
      ...
```

1 外部 IP アドレスのサービスへの自動割り当てに使用できる CIDR 形式で IP アドレスブロックを定義します。1 つの IP アドレス範囲のみが許可されます。

2 IP アドレスのサービスへの手動割り当ての制限を定義します。制限が定義されていない場合は、サービスに **spec.externalIP** フィールドを指定しても許可されません。デフォルトで、制限は定義されません。

以下の YAML は、**policy** スタンザのフィールドについて説明しています。

Network.config.openshift.io policy スタンザ

```
policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2
```

1 CIDR 形式の許可される IP アドレス範囲の一覧。

2 CIDR 形式の拒否される IP アドレス範囲の一覧。

外部 IP 設定の例

外部 IP アドレスプールの予想される複数の設定が以下の例で表示されています。

- 以下の YAML は、自動的に割り当てられた外部 IP アドレスを有効にする設定について説明しています。

  spec.externalIP.autoAssignCIDRs が設定された設定例

  ■

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
    - 192.168.132.254/29
```

- 以下の YAML は、許可された、および拒否された CIDR 範囲のポリシールールを設定します。

  spec.externalIP.policy が設定された設定例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    policy:
      allowedCIDRs:
      - 192.168.132.0/29
      - 192.168.132.8/29
      rejectedCIDRs:
      - 192.168.132.7/32
```

第14章 クラスターの外部 IP アドレスブロックの設定

クラスター管理者は、以下の ExternalIP を設定できます。

- サービスの **spec.clusterIP** フィールドを自動的に設定するために OpenShift Container Platform によって使用される ExternalIP アドレスブロック。
- IP アドレスを制限するポリシーオブジェクトはサービスの **spec.clusterIP** 配列に手動で割り当てられます。

前提条件

- OpenShift CLI (**oc**) のインストール。
- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。

手順

1. オプション: 現在の外部 IP 設定を表示するには、以下のコマンドを入力します。

```
$ oc describe networks.config cluster
```

2. 設定を編集するには、以下のコマンドを入力します。

```
$ oc edit networks.config cluster
```

3. 以下の例のように ExternalIP 設定を変更します。

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
```

1 **externalIP** スタンザの設定を指定します。

4. 更新された ExternalIP 設定を確認するには、以下のコマンドを入力します。

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

14.1. 次のステップ

- サービスの外部 IP を使用した ingress クラストラフィックの設定

14.2. INGRESS コントローラーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は Ingress コントローラーを使用します。

14.2.1. Ingress コントローラーおよびルートの使用

Ingress Operator は Ingress コントローラーおよびワイルドカード DNS を管理します。

Ingress コントローラーの使用は、OpenShift Container Platform クラスターへの外部アクセスを許可するための最も一般的な方法です。

Ingress コントローラーは外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するよう設定されます。これは SNI を使用する HTTP、HTTPS、および SNI を使用する TLS に制限されますが、これは SNI を使用する TLS で機能する Web アプリケーションやサービスには十分な設定です。

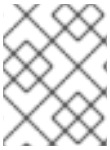
管理者と連携して Ingress コントローラーを設定します。外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するように Ingress コントローラーを設定します。

管理者はワイルドカード DNS エントリーを作成してから Ingress コントローラーを設定できます。その後は管理者に問い合わせることなく edge Ingress コントローラーと連携できます。

一連のルートが各種プロジェクトで作成される場合、ルートのセット全体が一連の Ingress コントローラーで利用可能になります。それぞれの Ingress コントローラーはルートのセットからのルートを許可します。デフォルトで、すべての Ingress コントローラーはすべてのルートを許可します。

Ingress コントローラー:

- デフォルトでは 2 つのレプリカがあるので、これは 2 つのワーカーノードで実行する必要があります。
- 追加のノードにレプリカを組み込むためにスケールアップすることができます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

14.2.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

14.2.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

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

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7   ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

14.2.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project project1
```

3. 以下のコマンドを実行してルートを公開します。

```
$ oc expose service <service_name>
```

以下は例になります。

```
$ oc expose service mysql-80-rhel7
```

出力例

```
route "mysql-80-rhel7" exposed
```

4. cURL などのツールを使用し、サービスのクラスター IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <pod_ip>:<port>
```

以下は例になります。

```
$ curl 172.30.131.89:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続されていることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

14.2.5. ルートラベルを使用した Ingress コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセレクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

```
# cat router-internal.yaml
apiVersion: v1
```

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

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

14.2.6. namespace ラベルを使用した Ingress コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セレクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

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

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
```

```
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    namespaceSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

14.2.7. 追加リソース

- Ingress Operator はワイルドカード DNS を管理します。詳細は、「OpenShift Container Platform の Ingress Operator」、「Installing a cluster on bare metal」、および「Installing a cluster on vSphere」を参照してください。

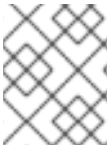
14.3. ロードバランサーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法では、ロードバランサーを使用します。

14.3.1. ロードバランサーを使用したトラフィックのクラスターへの送信

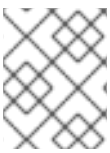
特定の外部 IP アドレスを必要としない場合、ロードバランサーサービスを OpenShift Container Platform クラスターへの外部アクセスを許可するよう設定することができます。

ロードバランサーサービスは固有の IP を割り当てます。ロードバランサーには単一の edge ルーター IP があります (これは仮想 IP (VIP) の場合もありますが、初期の負荷分散では単一マシンになります。



注記

プールが設定される場合、これはクラスター管理者によってではなく、インフラストラクチャーレベルで実行されます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

14.3.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

14.3.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

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

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7  ClusterIP  172.30.63.31   <none>        3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

14.3.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project project1
```

3. 以下のコマンドを実行してルートを公開します。

```
$ oc expose service <service_name>
```

以下は例になります。

```
$ oc expose service mysql-80-rhel7
```

出力例

```
route "mysql-80-rhel7" exposed
```

4. cURL などのツールを使用し、サービスのクラスター IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <pod_ip>:<port>
```

以下は例になります。

```
$ curl 172.30.131.89:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続されていることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
```

```
MySQL [(none)]>
```

14.3.5. ロードバランサーサービスの作成

以下の手順を使用して、ロードバランサーサービスを作成します。

前提条件

- 公開するプロジェクトとサービスがあること。

手順

ロードバランサーサービスを作成するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトを読み込みます。

```
$ oc project project1
```

3. マスターノードでテキストファイルを開き、以下のテキストを貼り付け、必要に応じてファイルを編集します。

ロードバランサー設定ファイルのサンプル

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 1
spec:
  ports:
  - name: db
    port: 3306 2
  loadBalancerIP:
  type: LoadBalancer 3
  selector:
    name: mysql 4
```

1. ロードバランサーサービスの説明となる名前を入力します。
2. 公開するサービスがリッスンしている同じポートを入力します。
3. タイプに **loadbalancer** を入力します。
4. サービスの名前を入力します。

4. ファイルを保存し、終了します。
5. 以下のコマンドを実行してサービスを作成します。

```
$ oc create -f <file-name>
```

以下は例になります。

```
$ oc create -f mysql-lb.yaml
```

6. 以下のコマンドを実行して新規サービスを表示します。

```
$ oc get svc
```

出力例

```
NAME      TYPE          CLUSTER-IP      EXTERNAL-IP                          PORT(S)
AGE
egress-2  LoadBalancer  172.30.22.226   ad42f5d8b303045-487804948.example.com
3306:30357/TCP  15m
```

有効にされたクラウドプロバイダーがある場合、サービスには外部 IP アドレスが自動的に割り当てられます。

7. マスターで cURL などのツールを使用し、パブリック IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <public-ip>:<port>
```

以下は例になります。

```
$ curl 172.29.121.74:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続していることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

14.4. サービスの外部 IP を使用した INGRESS クラスタートラフィックの設定

外部 IP アドレスをサービスに割り当てることで、これをクラスター外のトラフィックで使用できるようにします。通常、これはベアメタルハードウェアにインストールされているクラスターの場合にのみ役立ちます。外部ネットワークインフラストラクチャーは、トラフィックをサービスにルーティングするように正しく設定される必要があります。

14.4.1. 前提条件

- クラスターは ExternalIP が有効にされた状態で設定されます。詳細は、サービスの ExternalIP の設定について参照してください。

14.4.2. ExternalIP のサービスへの割り当て

ExternalIP をサービスに割り当てることができます。クラスターが ExternalIP を自動的に割り当てするように設定されている場合、ExternalIP をサービスに手動で割り当てる必要がない場合があります。

手順

1. オプション: ExternalIP で使用するために設定される IP アドレス範囲を確認するには、以下のコマンドを入力します。

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
```

   autoAssignCIDRs が設定されている場合、**spec.externalIPs** フィールドが指定されていない場合、 OpenShift Container Platform は ExternalIP を新規サービスに自動的に割り当てます。

2. ExternalIP をサービスリソースに割り当てます。

   a. 新規サービスを作成する場合は、**spec.externalIPs** フィールドを指定し、1 つ以上の有効な IP アドレスの配列を指定します。以下は例になります。

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externalIPs:
  - 192.174.120.10
```

   b. ExternalIP を既存のサービスに割り当てる場合は、以下のコマンドを入力します。**<name>** をサービス名に置き換えます。**<ip_address>** を有効な ExternalIP アドレスに置き換えます。コンマで区切られた複数の IP アドレスを指定できます。

```
$ oc patch svc <name> -p \
  '{
    "spec": {
      "externalIPs": [ "<ip_address>" ]
    }
  }'
```

   以下は例になります。

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

   出力例

```
"mysql-55-rhel7" patched
```

3. ExternalIP アドレスがサービスに割り当てられていることを確認するには、以下のコマンドを入力します。新規サービスに ExternalIP を指定した場合、まずサービスを作成する必要があります。

```
$ oc get svc
```

出力例

```
NAME            CLUSTER-IP      EXTERNAL-IP     PORT(S)    AGE
mysql-55-rhel7    172.30.131.89   192.174.120.10  3306/TCP   13m
```

14.4.3. 追加リソース

- サービスの ExternalIP の設定

14.5. NODEPORT を使用した INGRESS クラスタートラフィックの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は **NodePort** を使用します。

14.5.1. NodePort を使用したトラフィックのクラスターへの送信

NodePort-type **Service** リソースを使用して、クラスター内のすべてのノードの特定のポートでサービスを公開します。ポートは **Service** リソースの **.spec.ports[*].nodePort** フィールドで指定されます。

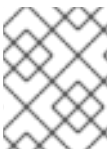
重要

NodePort を使用するには、追加のポートリソースが必要です。

NodePort は、ノードの IP アドレスの静的ポートでサービスを公開します。**NodePort** はデフォルトで **30000** から **32767** の範囲に置かれます。つまり、 **NodePort** はサービスの意図されるポートに一致しないことが予想されます。たとえば、ポート **8080** はノードのポート **31020** として公開できます。

管理者は、外部 IP アドレスがノードにルーティングされることを確認する必要があります。

NodePort および外部 IP は独立しており、両方を同時に使用できます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

14.5.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この

手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

14.5.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

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

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7  ClusterIP  172.30.63.31   <none>        3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

14.5.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project project1
```

3. アプリケーションのノードポートを公開するには、以下のコマンドを入力します。OpenShift Container Platform は **30000-32767** 範囲の利用可能なポートを自動的に選択します。

```
$ oc expose dc mysql-80-rhel7 --type=NodePort --name=mysql-ingress
```

4. オプション: サービスが公開されるノードポートで利用可能なことを確認するには、以下のコマンドを入力します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
mysql-80-rhel7  ClusterIP  172.30.217.127   <none>        3306/TCP         9m44s
mysql-ingress   NodePort   172.30.107.72    <none>        3306:31345/TCP   39s
```

5. オプション: **oc new-app** コマンドによって自動的に作成されたサービスを削除するには、以下のコマンドを入力します。

```
$ oc delete svc mysql-80-rhel7
```

第15章 クラスター全体のプロキシーの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシーを使用することができます。既存クラスターのプロキシーオブジェクトを変更するか、または新規クラスターの **install-config.yaml** ファイルでプロキシー設定を行うことにより、OpenShift Container Platform をプロキシーを使用するように設定できます。

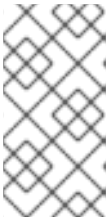
重要

クラスター全体のプロキシーは、ユーザーによってプロビジョニングされるインフラストラクチャーのインストールを使用している場合や、サポートされるプロバイダーに、仮想プライベートクラウドや仮想ネットワークなどの独自のネットワークを提供する場合にのみサポートされます。

15.1. 前提条件

- クラスターがアクセスする必要のあるサイトを確認します。

> and determine whether any of them must bypass the proxy. By default, all cluster egress traffic is proxied, including calls to the cloud provider API for the cloud that hosts your cluster. Add sites to the Proxy object's `spec.noProxy` field to bypass the proxy if necessary.



注記

プロキシーオブジェクトの **status.noProxy** フィールドは、デフォルトでインスタンスメタデータエンドポイント (**169.254.169.254**) およびインストール設定の **networking.machineCIDR**、 **networking.clusterNetwork.cidr**、および **networking.serviceNetwork[]** フィールドの値で設定されます。

15.2. クラスター全体のプロキシーの有効化

プロキシーオブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシーを設定せずにクラスターがインストールまたはアップグレードされると、プロキシーオブジェクトは引き続き生成されますが、**spec** は設定されません。以下は例になります。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

クラスター管理者は、この **cluster** プロキシーオブジェクトを変更して OpenShift Container Platform のプロキシーを設定できます。

注記

cluster という名前のプロキシーオブジェクトのみがサポートされ、追加のプロキシーは作成できません。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる ConfigMap を作成します。

注記

プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

```
apiVersion: v1
data:
  ca-bundle.crt: | (1)
    <MY_PEM_ENCODED_CERTS> (2)
kind: ConfigMap
metadata:
  name: user-ca-bundle (3)
  namespace: openshift-config (4)
```

(1) このデータキーは **ca-bundle.crt** という名前にする必要があります。

(2) プロキシーのアイデンティティー証明書に署名するために使用される 1 つ以上の PEM でエンコードされた X.509 証明書。

(3) プロキシーオブジェクトから参照される ConfigMap 名。

(4) ConfigMap は **openshift-config** namespace になければなりません。

b. このファイルから ConfigMap を作成します。

```
$ oc create -f user-ca-bundle.yaml
```

2. **oc edit** コマンドを使用してプロキシーオブジェクトを変更します。

```
$ oc edit proxy/cluster
```

3. プロキシーに必要なフィールドを設定します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> (1)
  httpsProxy: http://<username>:<pswd>@<ip>:<port> (2)
```

```
noProxy: example.com 3
readinessEndpoints:
- http://www.google.com 4
- https://www.google.com
trustedCA:
  name: user-ca-bundle 5
```

1. クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。
2. クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。これが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。
3. プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のカンマ区切りの一覧。ドメインのすべてのサブドメインを組み込むために、ドメインの前に **.** を入力します。* を使用し、すべての宛先のプロキシーをバイパスします。**networking.machineCIDR** に含まれていないワーカーをスケールアップする場合、 それらをこの一覧に追加し、接続の問題を防ぐ必要があります。
4. **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。
5. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の ConfigMap の参照。ここで参照する前に ConfigMap が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

15.3. クラスター全体のプロキシーの削除

cluster プロキシーオブジェクトは削除できません。クラスターからプロキシーを削除するには、プロキシーオブジェクトからすべての **spec** フィールドを削除します。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. **oc edit** コマンドを使用してプロキシーを変更します。

```
$ oc edit proxy/cluster
```

2. プロキシーオブジェクトからすべての **spec** フィールドを削除します。以下は例になります。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
status: {}
```

-

3. 変更を適用するためにファイルを保存します。

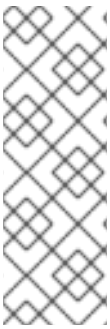
第16章 カスタム PKI の設定

Web コンソールなどの一部のプラットフォームコンポーネントは、通信にルートを使用し、それらと対話するために他のコンポーネントの証明書を信頼する必要があります。カスタムのパブリックキーインフラストラクチャー (PKI) を使用している場合は、プライベートに署名された CA 証明書がクラスター全体で認識されるようにこれを設定する必要があります。

プロキシー API を使用して、クラスター全体で信頼される CA 証明書を追加できます。インストール時またはランタイム時にこれを実行する必要があります。

- **インストール** 時に、クラスター全体のプロキシーを設定します。プライベートに署名された CA 証明書は、**install-config.yaml** ファイルの **additionalTrustBundle** 設定で定義する必要があります。
  インストールプログラムは、定義した追加の CA 証明書が含まれる **user-ca-bundle** という名前の ConfigMap を生成します。次に Cluster Network Operator は、これらの CA 証明書を Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** ConfigMap を作成し、この ConfigMap はプロキシーオブジェクトの **trustedCA** フィールドで参照されます。

- **ランタイム** 時に、デフォルトのプロキシーオブジェクトを変更して、プライベートに署名された CA 証明書を追加 します (これは、クラスターのプロキシー有効化のワークフローの一部です)。これには、クラスターで信頼される必要があるプライベートに署名された CA 証明書が含まれる ConfigMap を作成し、次にプライベートに署名された証明書の ConfigMap を参照する **trustedCA** でプロキシーリソースを変更することが関係します。



注記

インストーラー設定の **additionalTrustBundle** フィールドおよびプロキシーリソースの **trustedCA** フィールドは、クラスター全体の信頼バンドルを管理するために使用されます。 **additionalTrustBundle** はインストール時に使用され、プロキシーの **trustedCA** がランタイム時に使用されます。

trustedCA フィールドは、クラスターコンポーネントによって使用されるカスタム証明書とキーのペアを含む **ConfigMap** の参照です。

16.1. インストール時のクラスター全体のプロキシーの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシーを使用することができます。プロキシー設定を **install-config.yaml** ファイルで行うことにより、新規の OpenShift Container Platform クラスターをプロキシーを使用するように設定できます。

前提条件

- 既存の **install-config.yaml** ファイルが必要です。

- クラスターがアクセスする必要のあるサイトを確認し、プロキシーをバイパスする必要があるかどうかを判別します。デフォルトで、すべてのクラスター egress トラフィック (クラスターをホストするクラウドについてのクラウドプロバイダー API に対する呼び出しを含む) はプロキシーされます。プロキシーオブジェクトの **spec.noProxy** フィールドにサイトを追加し、必要に応じてプロキシーをバイパスします。

注記

プロキシーオブジェクトの **status.noProxy** フィールドは、デフォルトでインスタンスメタデータエンドポイント (**169.254.169.254**) およびインストール設定の **networking.machineCIDR**、 **networking.clusterNetwork.cidr**、および **networking.serviceNetwork[]** フィールドの値で設定されます。

手順

1. **install-config.yaml** ファイルを編集し、プロキシー設定を追加します。以下は例になります。

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
additionalTrustBundle: | 4
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

1 クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、**httpProxy** 値を指定することはできません。

2 クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。このフィールドが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、**httpsProxy** 値を指定することはできません。

3 プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のカンマ区切りの一覧。ドメインのすべてのサブドメインを組み込むために、ドメインの前に **.** を入力します。* を使用し、すべての宛先のプロキシーをバイパスします。

4 指定されている場合、インストールプログラムは HTTPS 接続のプロキシーに必要な 1 つ以上の追加の CA 証明書が含まれる **user-ca-bundle** という名前の ConfigMap を **openshift-config** namespace に生成します。次に Cluster Network Operator は、これらのコンテンツを Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** ConfigMap を作成し、この ConfigMap はプロキシーオブジェクトの **trustedCA** フィールドで参照されます。**additionalTrustBundle** フィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、MITM CA 証明書を指定する必要があります。

注記

インストールプログラムは、プロキシーの **readinessEndpoints** フィールドをサポートしません。

2. ファイルを保存し、OpenShift Container Platform のインストール時にこれを参照します。

インストールプログラムは、指定の **install-config.yaml** ファイルのプロキシー設定を使用する **cluster** という名前のクラスター全体のプロキシーを作成します。プロキシー設定が指定されていない場合、**cluster** のプロキシーオブジェクトが依然として作成されますが、これには **spec** がありません。

注記

cluster という名前のプロキシーオブジェクトのみがサポートされ、追加のプロキシーは作成できません。

16.2. クラスター全体のプロキシーの有効化

プロキシーオブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシーを設定せずにクラスターがインストールまたはアップグレードされると、プロキシーオブジェクトは引き続き生成されますが、**spec** は設定されません。以下は例になります。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

クラスター管理者は、この **cluster** プロキシーオブジェクトを変更して OpenShift Container Platform のプロキシーを設定できます。

注記

cluster という名前のプロキシーオブジェクトのみがサポートされ、追加のプロキシーは作成できません。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる ConfigMap を作成します。

注記

プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

```
apiVersion: v1
```

```
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

1. このデータキーは **ca-bundle.crt** という名前にする必要があります。
2. プロキシーのアイデンティティー証明書に署名するために使用される 1 つ以上の PEM でエンコードされた X.509 証明書。
3. プロキシーオブジェクトから参照される ConfigMap 名。
4. ConfigMap は **openshift-config** namespace になければなりません。

b. このファイルから ConfigMap を作成します。

```
$ oc create -f user-ca-bundle.yaml
```

2. **oc edit** コマンドを使用してプロキシーオブジェクトを変更します。

```
$ oc edit proxy/cluster
```

3. プロキシーに必要なフィールドを設定します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
  - http://www.google.com 4
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

1. クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。
2. クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。これが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。
3. プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のカンマ区切りの一覧。ドメインのすべてのサブドメインを組み込むために、ドメインの前に **.** を入力します。* を使用し、すべての宛先のプロキシーをバイパスします。**networking.machineCIDR** に含まれていないワーカーをスケールアップする場合、 それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

4 **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。

5 HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の ConfigMap の参照。ここで参照する前に ConfigMap が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

16.3. OPERATOR を使用した証明書の挿入

カスタム CA 証明書が ConfigMap 経由でクラスターに追加されると、Cluster Network Operator はユーザーによってプロビジョニングされる CA 証明書およびシステム CA 証明書を単一バンドルにマージし、信頼バンドルの挿入を要求する Operator にマージされたバンドルを挿入します。

Operator は、以下のラベルの付いた空の ConfigMap を作成してこの挿入を要求します。

```
config.openshift.io/inject-trusted-cabundle="true"
```

Operator は、この ConfigMap をコンテナーのローカル信頼ストアにマウントします。

注記

信頼された CA 証明書の追加は、証明書が Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルに含まれない場合にのみ必要になります。

証明書の挿入は Operator に制限されません。Cluster Network Operator は、空の ConfigMap が **config.openshift.io/inject-trusted-cabundle=true** ラベルを使用して作成される場合に、すべての namespace で証明書を挿入できます。

ConfigMap はすべての namespace に置くことができますが、ConfigMap はカスタム CA を必要とする Pod 内の各コンテナーに対してボリュームとしてマウントされる必要があります。以下は例になります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
```

```
    name: trusted-ca
    items:
      - key: ca-bundle.crt (1)
        path: tls-ca-bundle.pem (2)
```

(1) **ca-bundle.crt** は ConfigMap キーとして必要になります。

(2) **tls-ca-bundle.pem** は ConfigMap パスとして必要になります。