OpenShift Container Platform 4.1

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: 2020-05-15

OpenShift Container Platform 4.1 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。

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

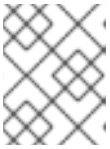
OpenShift Container Platform インスタンスにアクセスして、セキュアなシェル (SSH) アクセスでマスターノードにアクセスするために bastion ホストを作成する方法を学びます。

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) Software Defined Networking (SDN) 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 を使って OpenShift SDN プラグイン、またはクラスターインストール時に選択されている場合には別の SDN 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

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

NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.1.0     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

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
```

3.5. CLUSTER NETWORK OPERATOR のカスタムリソース (CR、CUSTOM RESOURCE)

Network.operator.openshift.io カスタムリソース (CR) のクラスターネットワーク設定は、Cluster Network Operator (CNO) の設定内容を保存します。

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
spec:
  clusterNetwork: (1)
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: (2)
  - 172.30.0.0/16
  defaultNetwork:
    type: OpenShiftSDN (3)
    openshiftSDNConfig: (4)
      mode: NetworkPolicy (5)
      mtu: 1450 (6)
      vxlanPort: 4789 (7)
  kubeProxyConfig: (8)
    iptablesSyncPeriod: 30s (9)
    proxyArguments:
      iptables-min-sync-period: (10)
      - 30s
```

(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) プラグインは、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) 使用されている Software Defined Networking (SDN) プラグイン。 OpenShift SDN は OpenShift Container Platform 4.1 でサポートされている唯一のプラグインです。

(4) OpenShift SDN 固有の設定パラメーター。

(5) OpenShift SDN CNI プラグインの分離モード。

(6) VXLAN 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。

(7) 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。

(8) 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。

(9) **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 time package ドキュメントで説明されています。

(10) **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** が含まれ、これらについては、Go time package で説明されています。

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
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
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。これは、カスタマイズしたり、置き換えたり、または追加の **dnses** で補足したりすることができません。

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
Name:         default
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         DNS
...
```

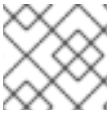
```
Status:
  Cluster Domain:  cluster.local 1
  Cluster IP:      172.30.0.10 2
...
```

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}'
[172.30.0.0/16]
```



注記

CoreDNS Corefile または Kubernetes プラグインの設定はサポートされていません。

4.3. 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
```

4.4. 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 --namespace=openshift-dns-operator deployment/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`

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によって署名され、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
NAME      AGE
default   10m
```

警告

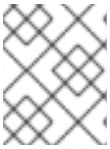
デフォルトの証明書が置き換えられる場合、コンテナーのユーザー空間で提供される、CA バンドルにすでに含まれているパブリック認証局による署名が **必要** になります。

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** 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}'
```

出力は以下のようになります。

```
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 Controller 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。

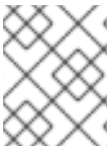
```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
2
```

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
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}'
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
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。

第6章 複数ネットワークの管理

6.1. 概要

Multus CNI は、OpenShift Container Platform で複数のネットワークインターフェースを Pod に割り当てる機能を提供します。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 を設定する必要がある場合に柔軟性を実現します。

Multus CNI は、データプレーンとコントロールプレーンの分離などの、ネットワークの分離が必要な状況で役立ちます。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の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 を使用してネットワークインターフェースを追加する場合、それらの名前は **net1**、**net2**、... になります。**netN** などになります。

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) を使用して指定されます。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。Multus CNI は、他の CNI プラグインを呼び出すことのできる CNI プラグインです。これにより、他の CNI プラグインを使用して追加のネットワークインターフェースを作成できます。高パフォーマンスのネットワークを達成するには、Multus CNI で SR-IOV デバイスプラグインを使用します。

以下の手順を実行して、追加のネットワークインターフェースを Pod に割り当てます。

1. CNI 設定をカスタムリソースとして作成します。
2. Pod に設定名のアノテーションを付けます。
3. ステータスアノテーションを表示して、割り当てが正常に行われたことを確認します。

6.2. CNI 設定

CNI は、**type** という 1 つの必須フィールドのみを持つ JSON データです。**additional** フィールドの設定は、フリーフォームの JSON データです。これにより、CNI プラグインを使用し、必要なフォームで設定を行うことができます。異なる CNI プラグインが異なる設定を使用します。使用する必要のある CNI プラグイン専用のドキュメントを参照してください。

CNI 設定例:

```
{
  "cniVersion": "0.3.0", 1
  "type": "loopback", 2
```

```
  "additional": "<plugin-specific-json-data>" 3
}
```

1. **cniVersion**: 使用される CNI バージョンを指定します。CNI プラグインはこの情報を使用して、有効なバージョンを使用しているかどうかをチェックします。

2. **type**: ディスクで呼び出す CNI プラグインバイナリーを指定します。この例では、loopback バイナリーが指定されています。 そのため、これは loopback タイプのネットワークインターフェースを作成します。

3. **additional**: 上記のコードで提供される **<information>** 値は一例になります。各 CNI プラグインは JSON で必要な設定パラメーターを指定します。これらは、 **type** フィールドの名前が付けられた CNI プラグインバイナリーに固有のパラメーターです。

6.3. 追加のネットワークインターフェースの作成

Pod の追加インターフェースは、カスタムリソース (CR) として保存される CNI 設定で定義されます。これらの CR は、**oc** ツールを使用して作成し、一覧表示し、編集し、削除できます。

以下の手順は、Pod に **macvlan** インターフェースを設定します。この設定は、すべての実稼働環境に適用されない可能性がありますが、他の CNI プラグインに同じ手順を使用できます。

6.3.1. 追加インターフェースの CNI 設定の CR としての作成

注記

追加インターフェースを Pod に割り当てる必要がある場合、インターフェースを定義する CR は Pod と同じプロジェクト (namespace) に置かれる必要があります。

1. CNI 設定を CR として保存するプロジェクト、およびその CR を使用する Pod を作成します。

```
$ oc new-project multinetwork-example
```

2. 追加のネットワークインターフェースを定義する CR を作成します。以下の内容を含む、**macvlan-conf.yaml** という YAML ファイルを作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition 1
metadata:
  name: macvlan-conf 2
spec:
  config: '{ 3
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth0",
      "mode": "bridge",
      "ipam": {
        "type": "host-local",
        "subnet": "192.168.1.0/24",
        "rangeStart": "192.168.1.200",
        "rangeEnd": "192.168.1.216",
        "routes": [
```

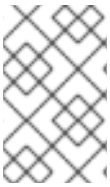
```
        { "dst": "0.0.0.0/0" }
      ],
      "gateway": "192.168.1.1"
    }
}'
```

1. **kind: NetworkAttachmentDefinition**: これは、この設定が保存される CR の名前です。これは、この設定が保存される CR の名前です。これは、ネットワークを Pod に割り当てる方法を定義する Kubernetes のカスタム拡張です。
2. **name** は、次の手順で使用されるアノテーションにマップされます。
3. **config**: CNI 設定は **config** フィールドにパッケージ化されます。

設定はプラグインに固有のものです。CNI 設定部分の **type** 行に注意してください。CNI 設定の部分の **type** 行をメモしてください。 この例では、ネットワーク用の IPAM (IP アドレス管理) パラメーターのほかに、**master** フィールドは、Pod をホストするノードにあるネットワークインターフェースを参照する必要があります。

3. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f macvlan-conf.yaml
```



注記

この例は、**macvlan** CNI プラグインをベースにしています。AWS 環境では、macvlan トラフィックはフィルターされる可能性があるため、必要な宛先に到達しない可能性があります。

6.3.2. 追加インターフェースの CR の管理

追加インターフェースの CR は、**oc** CLI を使用して管理できます。

以下のコマンドを使用して、追加リソースの CR を一覧表示します。

```
$ oc get network-attachment-definitions.k8s.cni.cncf.io
```

以下のコマンドを使用して、追加インターフェースの CR を削除します。

```
$ oc delete network-attachment-definitions.k8s.cni.cncf.io macvlan-conf
```

6.3.3. CR を使用するアノテーション付き Pod の作成

追加インターフェースを使用する Pod を作成するには、CR を参照するアノテーションを使用します。以下の内容を含む Pod についての **samplepod.yaml** という YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf 1
spec:
```

```
  containers:
  - name: samplepod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

1 **annotations** フィールドには、**k8s.v1.cni.cncf.io/networks: macvlan-conf** が含まれます。 これは、先に定義した CR の **name** フィールドに相関します。

以下のコマンドを実行して **samplepod** Pod を作成します。

```
$ oc create -f samplepod.yaml
```

追加のネットワークインターフェースが作成され、Pod に割り当てられているのを確認するには、以下のコマンドを使用して IPv4 アドレス情報を一覧表示します。

```
$ oc exec -it samplepod -- ip -4 addr
```

3 つのインターフェースが出力に一覧表示されます。

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000 1
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP  link-
netnsid 0 2
    inet 10.244.1.4/24 scope global eth0
       valid_lft forever preferred_lft forever
4: net1@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN  link-netnsid 0 3
    inet 192.168.1.203/24 scope global net1
       valid_lft forever preferred_lft forever
```

1 **lo**: ループバックインターフェースです。

2 **eth0**: クラスター全体のデフォルトネットワークに接続されるインターフェースです。

3 **net1**: 作成したばかりの新規インターフェースです。

6.3.3.1. 複数インターフェースの Pod への割り当て

複数の追加インターフェースを Pod に割り当てるには、複数の名前を、Pod 定義の **annotations** フィールドにカンマ区切りの形式で指定します。

Pod 定義の以下の **annotations** フィールドは、追加インターフェースの複数の異なる CR を指定します。

```
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf, tertiary-conf, quaternary-conf
```

Pod 定義の以下の **annotations** フィールドは、追加インターフェースに同じ CR を指定します。

```
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf, macvlan-conf
```

6.3.4. 実行中の Pod のインターフェース設定の表示

Pod の実行後に、作成された追加インターフェースの設定を確認できます。先の例と同じ Pod を表示するには、以下のコマンドを実行します。

```
$ oc describe pod samplepod
```

出力の **metadata** セクションには、JSON 形式で表示されるアノテーションの一覧が含まれます。

```
Annotations:
  k8s.v1.cni.cncf.io/networks: macvlan-conf
  k8s.v1.cni.cncf.io/networks-status:
   [{
       "name": "openshift-sdn",
       "ips": [
           "10.131.0.10"
       ],
       "default": true,
       "dns": {}
   },{
       "name": "macvlan-conf", (1)
       "interface": "net1", (2)
       "ips": [ (3)
           "192.168.1.200"
       ],
       "mac": "72:00:53:b4:48:c4", (4)
       "dns": {} (5)
   }]
```

1. **name** は、カスタムリソース名 **macvlan-conf** を参照します。
2. **interface** は、Pod のインターフェースの名前を参照します。
3. **ips** は、Pod に割り当てられる IP アドレスの一覧です。
4. **mac** は、インターフェースの MAC アドレスです。
5. **dns** は、インターフェースの DNS を参照します。

最初のアノテーション **k8s.v1.cni.cncf.io/networks: macvlan-conf** は、例で作成された CR を参照します。このアノテーションは Pod 定義で指定されています。

2 つ目のアノテーションは、 **k8s.v1.cni.cncf.io/networks-status** です。**k8s.v1.cni.cncf.io/networks-status** には 2 つのインターフェースが一覧表示されます。

- 最初のインターフェースは、デフォルトネットワーク **openshift-sdn** のインターフェースを記述します。このインターフェースは **eth0** として作成されます。これは、クラスター内の通信に使用されます。
- 2 つ目のインターフェースは、作成した追加インターフェース **net1** です。上記の出力は、Pod に割り当てられた IP アドレスなど、インターフェースの作成時に設定されたいくつかのキーの値を一覧表示しています。

6.4. ホストデバイスを使用した追加インターフェースの設定

ホストデバイスプラグインは、既存のネットワークデバイスを Pod に直接接続します。

以下のコードは、ダミーモジュールを使用してダミーデバイスを作成し、仮想デバイスをサポートし、ダミーデバイス **name** を **exampledevice0** に割り当てます。

```
$ modprobe dummy
$ lsmod | grep dummy
$ ip link add exampledevice0 type dummy
```

手順

1. ダミーネットワークデバイスを Pod に接続するには、Pod をデバイスがあるノードに割り当てられるようホストにラベルを付けます。

```
$ oc label nodes <your-worker-node-name> exampledevice=true
$ oc get nodes --show-labels
```

2. この設定を参照するために、カスタムリソースの **hostdevice-example.yaml** という YAML ファイルを作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: hostdevice-example
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "host-device",
      "device": "exampledevice0"
    }'
```

3. 以下のコマンドを実行して **hostdevice-example** CR を作成します。

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

4. アノテーションでこの名前を参照する Pod の YAML ファイルを作成します。Pod をエイリアスを作成したマシンに割り当てるために **nodeSelector** を組み込みます。

```
apiVersion: v1
kind: Pod
metadata:
  name: hostdevicesamplepod
  annotations:
    k8s.v1.cni.cncf.io/networks: hostdevice-example
spec:
  containers:
  - name: hostdevicesamplepod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
  nodeSelector:
    exampledevice: "true"
```

5. 以下のコマンドを実行して **hostdevicesamplepod** Pod を作成します。

```
$ oc create -f hostdevicesamplepod.yaml
```

6. 作成した追加インターフェースを表示します。

```
$ oc exec hostdevicesamplepod -- ip a
```

重要

SR-IOV 複数ネットワークサポート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、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

6.5. SR-IOV の設定

OpenShift Container Platform には、OpenShift Container Platform ノードで SR-IOV ハードウェアを使用する機能が含まれます。これにより、他のネットワークインターフェースに加えて SR-IOV Virtual Function (VF) インターフェースを Pod に割り当てることができます。

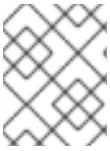
この機能を有効にするには、SR-IOV ネットワークデバイスプラグインおよび SR-IOV CNI プラグインという 2 つのコンポーネントが必要になります。

- SR-IOV ネットワークプラグインは、SR-IOV ネットワークの Virtual Function (VF) リソースを検出し、公開し、割り当てるための Kubernetes デバイスプラグインです。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 スケジューラーに使い切られたリソースについて認識させ、Pod が利用可能なリソースが十分にあるワーカーノードにスケジュールされるようにします。
- SR-IOV CNI プラグインは、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 Pod につなぎます。

6.5.1. サポートされるデバイス

以下のネットワークインターフェースカード (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)



注記

Mellanox カードの場合、VF をホストにプロビジョニングする前に SR-IOV がファームウェアで有効にされていることを確認します。

6.5.2. SR-IOV プラグインおよび daemonset の作成

注記

SR-IOV VF の作成は、SR-IOV デバイスプラグインおよび SR-IOV CNI では処理されません。SR-IOV VF をホストにプロビジョニングするには、これを手動で設定する必要があります。

SR-IOV ネットワークプラグインおよび SR-IOV CNI プラグインを使用するには、クラスター内の各ノード上で、両方のプラグインをデーモンモードで実行します。

1. 以下の内容を含む **openshift-sriov** namespace の YAML ファイルを作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov
  labels:
    name: openshift-sriov
    openshift.io/run-level: "0"
  annotations:
    openshift.io/node-selector: ""
    openshift.io/description: "Openshift SR-IOV network components"
```

2. 以下のコマンドを実行して、**openshift-sriov** namespace を作成します。

```
$ oc create -f openshift-sriov.yaml
```

3. 以下の内容を含む **sriov-device-plugin** サービスアカウントの YAML ファイルを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sriov-device-plugin
  namespace: openshift-sriov
```

4. 以下のコマンドを実行し、**sriov-device-plugin** サービスアカウントを作成します。

```
$ oc create -f sriov-device-plugin.yaml
```

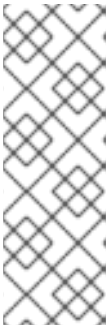
5. 以下の内容を含む **sriov-cni** サービスアカウントの YAML ファイルを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sriov-cni
  namespace: openshift-sriov
```

6. 以下のコマンドを実行し、**sriov-cni** サービスアカウントを作成します。

```
$ oc create -f sriov-cni.yaml
```

7. 以下の内容を含む **sriov-device-plugin** DaemonSet の YAML ファイルを作成します。



注記

SR-IOV ネットワークデバイスプラグインデーモンは、起動時に各ホストで設定されたすべての (サポートされている NIC モデルの) SR-IOV VF を検出し、検出されたリソースを公開します。割り当てることのできる利用可能な SR-IOV VF リソースの数は、**oc describe node <node-name>** コマンドでノードを記述して確認できます。SR-IOV VF リソースのリソース名は **openshift.io/sriov** です。ノードで利用可能な SR-IOV VF がない場合、ゼロ (0) の値が表示されます。

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: sriov-device-plugin
  namespace: openshift-sriov
  annotations:
    kubernetes.io/description: |
      This daemon set launches the SR-IOV network device plugin on each node.
spec:
  selector:
    matchLabels:
      app: sriov-device-plugin
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: sriov-device-plugin
        component: network
        type: infra
        openshift.io/component: network
    spec:
      hostNetwork: true
      nodeSelector:
        beta.kubernetes.io/os: linux
      tolerations:
      - operator: Exists
      serviceAccountName: sriov-device-plugin
      containers:
      - name: sriov-device-plugin
        image: quay.io/openshift/ose-sriov-network-device-plugin:v4.0.0
        args:
        - --log-level=10
        securityContext:
          privileged: true
        volumeMounts:
        - name: devicesock
          mountPath: /var/lib/kubelet/
          readOnly: false
```

```
        - name: net
          mountPath: /sys/class/net
          readOnly: true
      volumes:
        - name: devicesock
          hostPath:
            path: /var/lib/kubelet/
        - name: net
          hostPath:
            path: /sys/class/net
```

8. 以下のコマンドを実行し、**sriov-device-plugin** DaemonSet を作成します。

```
oc create -f sriov-device-plugin.yaml
```

9. 以下の内容を含む **sriov-cni** DaemonSet の YAML ファイルを作成します。

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: sriov-cni
  namespace: openshift-sriov
  annotations:
    kubernetes.io/description: |
      This daemon set launches the SR-IOV CNI plugin on SR-IOV capable worker nodes.
spec:
  selector:
    matchLabels:
      app: sriov-cni
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: sriov-cni
        component: network
        type: infra
        openshift.io/component: network
    spec:
      nodeSelector:
        beta.kubernetes.io/os: linux
      tolerations:
      - operator: Exists
      serviceAccountName: sriov-cni
      containers:
      - name: sriov-cni
        image: quay.io/openshift/ose-sriov-cni:v4.0.0
        securityContext:
          privileged: true
        volumeMounts:
        - name: cnibin
          mountPath: /host/opt/cni/bin
      volumes:
```

```
        - name: cnibin
          hostPath:
            path: /var/lib/cni/bin
```

10. 以下のコマンドを実行し、**sriov-cni** DaemonSet を作成します。

```
$ oc create -f sriov-cni.yaml
```

6.5.3. SR-IOV を使用した追加インターフェースの設定

1. SR-IOV 設定を使用してカスタムリソース (CR) の YAML ファイルを作成します。以下の CR の **name** フィールドには、値 **sriov-conf** が含まれます。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-conf
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/sriov (1)
spec:
  config: '{
      "type": "sriov", (2)
      "name": "sriov-conf",
      "ipam": {
        "type": "host-local",
        "subnet": "10.56.217.0/24",
        "routes": [{
          "dst": "0.0.0.0/0"
        }],
        "gateway": "10.56.217.1"
      }
    }'
```

(1) **k8s.v1.cni.cncf.io/resourceName** アノテーションは **openshift.io/sriov** に設定されます。

(2) **type** は **sriov** に設定されます。

2. 以下のコマンドを実行して、**sriov-conf** CR を作成します。

```
$ oc create -f sriov-conf.yaml
```

3. **NetworkAttachmentDefinition** の名前を参照し、1 つの **openshift.io/sriov** リソースを参照する Pod の YAML ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: sriovsamplepod
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-conf
spec:
  containers:
```

```
  - name: sriovsamplepod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
    resources:
      requests:
        openshift.io/sriov: '1'
      limits:
        openshift.io/sriov: '1'
```

4. 以下のコマンドを実行して **sriovsamplepod** Pod を作成します。

```
$ oc create -f sriovsamplepod.yaml
```

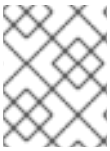
5. **ip** コマンドを実行して、追加のインターフェースを表示します。

```
$ oc exec sriovsamplepod -- ip a
```

第7章 OPENSHIFT SDN を使用したネットワークポリシーの設定

7.1. 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 4.1 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。

7.2. 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。

7.3. 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

- **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインを使用するクラスター。このモードは OpenShiftSDN のデフォルトです。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) をインストールします。
- 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。 ここで、**<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
networkpolicy "default-deny" created
```

7.4. 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

- **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインを使用するクラスター。このモードは OpenShiftSDN のデフォルトです。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) をインストールします。
- クラスターにログインする必要があります。

手順

- NetworkPolicy オブジェクトを削除するには、以下のコマンドを実行します。

```
$ oc delete networkpolicy -l name=<policy-name> 1
```

1 削除する NetworkPolicy オブジェクトの名前を指定します。

7.5. 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

- **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインを使用するクラスター。このモードは OpenShiftSDN のデフォルトです。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) をインストールします。
- 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
```

7.6. 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

- **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインを使用するクラスター。このモードは OpenShiftSDN のデフォルトです。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) をインストールします。

- クラスターにログインする必要があります。

手順

1. NetworkPolicy オブジェクト定義が含まれる以下のファイルを作成します。

   a. 以下を含む **allow-from-openshift-ingress.yaml** という名前のファイル。

```
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
```

   b. 以下を含む **allow-from-openshift-monitoring.yaml** という名前のファイル。

```
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
```

2. 各ポリシーファイルについて、以下のコマンドを実行し、NetworkPolicy オブジェクトを作成します。

```
$ oc apply -f <policy-name>.yaml \ 1
  -n <project> 2
```

   1 **<policy-name>** を、ポリシーを含むファイルのファイル名に置き換えます。

   2 **<project>** を NetworkPolicy オブジェクトを適用するプロジェクトの名前に置き換えます。

3. **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'
```

4. オプション: 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 networkpolicy allow-from-openshift-ingress -o yaml

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
```

7.7. 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。

7.7.1. 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。

7.7.2. 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

- **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインを使用するクラスター。このモードは OpenShiftSDN のデフォルトです。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

- **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
```

第8章 OPENSHIFT SDN

8.1. OPENSHIFT SDN 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 4.1 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。

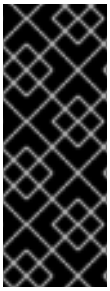
8.2. プロジェクトの EGRESS IP の設定

クラスター管理者として、OpenShift SDN ネットワークを 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。

8.2.1. 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。

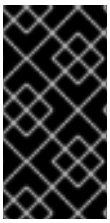
一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェースで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

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。

8.2.1.1. 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。

8.2.1.2. 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。

8.2.2. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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. 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。

8.2.3. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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. 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 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** 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。

8.3. 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。

8.3.1. egress ファイアウォールのプロジェクトでの機能

OpenShift Container Platform 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。

注意

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。

8.3.1.1. 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。
- **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 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。

8.3.1.2. 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。

8.3.1.3. 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。

8.3.2. 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。

```
kind: EgressNetworkPolicy
apiVersion: v1
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。

8.3.2.1. 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。

8.3.2.2. 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。

```
kind: EgressNetworkPolicy
apiVersion: v1
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。

8.3.3. egress ファイアウォールポリシーオブジェクトの作成

OpenShift Container Platform 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 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。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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
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。

8.4. 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。

8.4.1. EgressNetworkPolicy オブジェクトの編集

OpenShift Container Platform 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。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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. オプションとして、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
```

8.4.2. 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。

```
kind: EgressNetworkPolicy
apiVersion: v1
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。

8.4.2.1. 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。

8.4.2.2. 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。

```
kind: EgressNetworkPolicy
apiVersion: v1
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。

8.5. 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。

8.5.1. EgressNetworkPolicy オブジェクトの削除

OpenShift Container Platform 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。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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
```

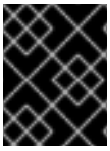
2. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを削除します。**<project>** をプロジェクトの名前に、 **<name>** をオブジェクトの名前に置き換えます。

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

8.6. マルチキャストの使用

8.6.1. 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 ネットワーク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。

8.6.2. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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**。

8.6.3. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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**。

8.7. 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。

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。

8.7.1. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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。

8.7.2. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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。

8.7.3. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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。

8.8. 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。

8.8.1. 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。

8.8.2. 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

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) 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
```

コマンドは、以下の例のような出力を返します。

```
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
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。

8.8.3. 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。

表8.1 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 time package 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。	**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 time package で説明されています。	**30s**

第9章 ルートの作成

9.1. ルート設定

9.1.1. 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。

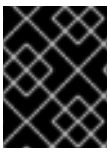
以下の例では、2 秒のタイムアウトを **myroute** という名前のルートに設定します。

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

9.1.2. 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。

9.1.3. 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ノードで tcpdump ツールを実行します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。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。

9.1.4. 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。

9.1.4.1. 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
```

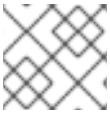
9.1.5. ルート固有のアノテーション

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9.1 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。	passthrough 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。

9.2. セキュリティー保護されたルート

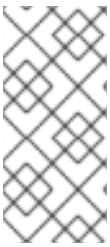
以下のセクションでは、カスタム証明書を使用して re-encrypt および edge ルートを作成する方法を説明します。

9.2.1. 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-----
```

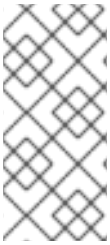
他のオプションについては、**oc create route reencrypt --help** を参照してください。

9.2.2. 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10章 INGRESS クラスタートラフィックの設定

10.1. 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。

10.2. 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。

10.2.1. 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。

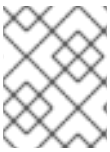
管理者と連携して Ingress コントローラーを設定します。外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するように Ingress コントローラーを設定します。

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。

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。

10.2.2. 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>
```

例:

```
$ 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
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。

10.2.3. 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>
```

例:

```
$ oc expose service mysql-80-rhel7
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>
```

例:

```
$ 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
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

10.2.4. 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。

10.2.5. 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
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。

10.2.6. 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。

10.3. 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。

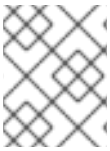
10.3.1. ロードバランサーを使用したトラフィックのクラスターへの送信

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。

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。

10.3.2. 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>
```

例:

```
$ 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
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。

10.3.3. 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>
```

例:

```
$ oc expose service mysql-80-rhel7
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>
```

例:

```
$ 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
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

10.3.4. 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>
```

例:

```
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
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>
```

例:

```
$ 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
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

10.4. 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

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、サービスの外部 IP を使用します。

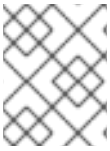
10.4.1. サービスの外部 IP を使用したトラフィックのクラスターへの送信

サービスを公開する 1 つの方法として、外部 IP アドレスをクラスター外からアクセス可能にするサービスに直接割り当てることができます。

使用する外部IPアドレスは、インフラストラクチャープラットフォームでプロビジョニングされ、クラスターノードに接続されている必要があります。

サービスに外部 IP を設定することにより、OpenShift Container Platform は、その IP アドレスに割り当てられるクラスターノードに到達するトラフィックが内部 Pod のいずれかに送信されることを許可する NAT ルールをセットアップします。これは内部サービス IP アドレスと似ていますが、外部 IP は OpenShift Container Platform に対し、このサービスが所定の IP で外部に公開される必要があることを示します。管理者は、この IP アドレスをクラスター内のノードのいずれかのホスト (ノード) インターフェースに割り当てる必要があります。または、このアドレスは仮想 IP (VIP) として使用することができます。

OpenShift Container Platform ではこれらの IP を管理しないため、管理者はトラフィックがこの IP を持つノードに到達することを確認する必要があります。



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。

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。

10.4.2. 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>
```

例:

```
$ 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
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。

10.4.3. 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>
```

例:

```
$ oc expose service mysql-80-rhel7
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>
```

例:

```
$ 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
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

10.5. 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。

10.5.1. 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。

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。

10.5.2. 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>
```

例:

```
$ 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
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。

10.5.3. 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
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
```