OpenShift Dedicated 4

ネットワーク

OpenShift Dedicated 4 でのネットワークの設定および管理

Last Updated: 2020-05-15

OpenShift Dedicated 4 ネットワーク

OpenShift Dedicated 4 でのネットワークの設定および管理

法律上の通知

概要

本書では、クラスター内のネットワーク設定の手順について説明します。

目次

第1章 ネットワークについて 3
1.1. OPENSHIFT DEDICATED DNS 3

第2章 ホストへのアクセス 4
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 4

第3章 OPENSHIFT DEDICATED の DNS OPERATOR 5
3.1. DNS OPERATOR 5
3.2. デフォルト DNS の表示 5
3.3. DNS 転送の使用 5

第4章 OPENSHIFT SDN を使用したネットワークポリシーの設定 8
4.1. ネットワークポリシーについて 8
4.2. サンプル NETWORKPOLICY オブジェクト 10
4.3. NETWORKPOLICY オブジェクトの作成 11
4.4. NETWORKPOLICY オブジェクトの削除 12
4.5. NETWORKPOLICY オブジェクトの表示 12
4.6. NETWORKPOLICY を使用したマルチテナント分離の設定 12

第5章 OPENSHIFT SDN ネットワークプロバイダー 15
5.1. OPENSHIFT SDN について 15
5.2. 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 15
5.3. プロジェクトの EGRESS ファイアウォールの編集 19
5.4. プロジェクトからの EGRESS ファイアウォールの削除 21

第6章 ルートの作成 22
6.1. ルート設定 22
6.2. セキュリティー保護されたルート 26

第7章 INGRESS クラスタートラフィックの設定 30
7.1. INGRESS コントローラーを使用した INGRESS クラスターの設定 30

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 DEDICATED 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 Dedicated 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

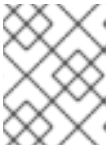
OpenShift Dedicated インスタンスにアクセスして、セキュアなシェル (SSH) アクセスでマスターノードにアクセスするために bastion ホストを作成する方法を学びます。

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 Dedicated インストーラーは、OpenShift Dedicated 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 Dedicated 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 Dedicated 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 Dedicated 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 Dedicated 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 DEDICATED 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。

3.1. 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。

3.2. デフォルト DNS の表示

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

手順

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

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

3.3. DNS 転送の使用

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

手順

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

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

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

DNS の例

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: foo-server 1
    zones: 2
      - foo.com
    forwardPlugin:
      upstreams: 3
        - 1.1.1.1
        - 2.2.2.2:5353
  - name: bar-server
    zones:
      - bar.com
      - example.com
    forwardPlugin:
      upstreams:
        - 3.3.3.3
        - 4.4.4.4:5454
```

1. **name** は、**rfc6335** サービス名の構文に準拠する必要があります。
2. **zones** は、**rfc1123** の **subdomain** の定義に準拠する必要があります。クラスタードメインの **cluster.local** は、 **zones** の無効な **subdomain** です。
3. **forwardPlugin** ごとに最大 15 の **upstreams** が許可されます。

注記

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

2. ConfigMap を表示します。

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

以前のサンプル DNS に基づく DNS ConfigMap の例

```
apiVersion: v1
data:
  Corefile: |
    foo.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
```

```
        forward . 3.3.3.3 4.4.4.4:5454 1
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            policy sequential
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

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

追加リソース

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

第4章 OPENSHIFT SDN を使用したネットワークポリシーの設定

4.1. 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 Dedicated 4.3 では、OpenShift SDN はデフォルトのネットワーク分離モードでの NetworkPolicy の使用をサポートしています。

注記

Kubernetes **v1** NetworkPolicy 機能は、Egress ポリシータイプおよび IPBlock 以外は OpenShift Dedicated 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 Dedicated 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 Dedicated 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。

4.2. サンプル NETWORKPOLICY オブジェクト

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

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy オブジェクトの **name**。
2. ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
3. ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。
4. トラフィックを受け入れる 1 つ以上の宛先の一覧。

4.3. 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
```

4.4. 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。

4.5. 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
```

4.6. 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 Dedicated 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
```

第5章 OPENSHIFT SDN ネットワークプロバイダー

5.1. OPENSHIFT SDN について

OpenShift Dedicated 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 Dedicated 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 は**ネットワークポリシー**モードのみをサポートします。これにより、プロジェクト管理者は NetworkPolicy オブジェクトを使用して独自の分離ポリシーを設定できます。

5.2. 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 Dedicated 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。

5.2.1. egress ファイアウォールのプロジェクトでの機能

クラスター管理者は、 **egress ファイアウォール** を使用して、一部またはすべての Pod がクラスター内からアクセスできる外部ホストを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Dedicated 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 Dedicated 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。

5.2.1.1. 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。

5.2.1.2. 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。

5.2.1.3. 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。

5.2.2. 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。

5.2.2.1. 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。

5.2.2.2. 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。

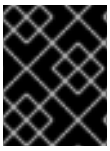
```
kind: EgressNetworkPolicy
apiVersion: v1
metadata:
  name: default-rules 1
spec:
  egress: 2
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

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。

5.2.3. egress ファイアウォールポリシーオブジェクトの作成

クラスター管理者は、プロジェクトの egress ファイアウォールポリシーオブジェクトを作成できます。



重要

プロジェクトに EgressNetworkPolicy オブジェクトがすでに定義されている場合、既存のポリシーを編集して egress ファイアウォールルールを変更する必要があります。

前提条件

- OpenShift SDN 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。

5.3. 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。

5.3.1. EgressNetworkPolicy オブジェクトの編集

クラスター管理者は、プロジェクトの egress ファイアウォールを更新できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- **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
```

5.3.2. 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。

5.3.2.1. 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。

5.3.2.2. 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。

5.4. 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 Dedicated 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。

5.4.1. EgressNetworkPolicy オブジェクトの削除

クラスター管理者は、プロジェクトから Egress ファイアウォールを削除できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- **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>
```

第6章 ルートの作成

6.1. ルート設定

6.1.1. 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。

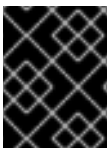
以下の例では、2 秒のタイムアウトを **myroute** という名前のルートに設定します。

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

6.1.2. 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。

6.1.3. スループット関連の問題のトラブルシューティング

OpenShift Dedicated 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 Dedicated 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。

6.1.4. Cookie に使用によるルートのステートフル性の維持

OpenShift Dedicated 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 Dedicated 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。

6.1.4.1. 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
```

6.1.5. 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6.1 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

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。	

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-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。	
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。

6.2. セキュリティー保護されたルート

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

重要

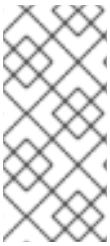
パブリックエンドポイントを使用して Microsoft Azure にルートを作成する場合、リソース名は制限されます。特定の用語を使用するリソースを作成することはできません。Azure が制限する語の一覧は、Azure ドキュメントの「Resolve reserved resource name errors」 を参照してください。

6.2.1. 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-----
```

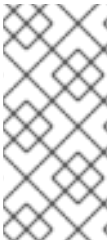
他のオプションについては、**oc create route reencrypt --help** を参照してください。

6.2.2. 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7章 INGRESS クラスタートラフィックの設定

7.1. INGRESS コントローラーを使用した INGRESS クラスターの設定

OpenShift Dedicated 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。

7.1.1. 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。

7.1.2. 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 Dedicated 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)]>
```