OpenShift Dedicated 4

ネットワーク

OpenShift Dedicated ネットワークの設定

Last Updated: 2024-10-15

OpenShift Dedicated 4 ネットワーク

OpenShift Dedicated ネットワークの設定

法律上の通知

概要

このドキュメントでは、OpenShift Dedicated クラスターのネットワークに関する情報を提供します。

目次

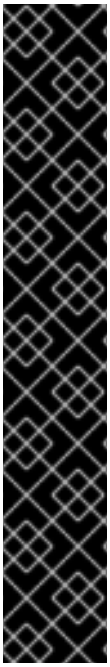
第1章 ネットワークの概要

Red Hat OpenShift Networking は、複数の機能、プラグイン、および高度なネットワーク機能からなるエコシステムです。これらの機能は、1 つまたは複数のハイブリッドクラスターのネットワークトラフィックを管理するためにクラスターに必要な高度なネットワーク関連機能により、Kubernetes ネットワークを強化します。このネットワーク機能のエコシステムは、Ingress、Egress、負荷分散、高性能スループット、セキュリティー、およびクラスター間およびクラスター内のトラフィック管理を統合します。また、Red Hat OpenShift Networking のエコシステムは、その固有の複雑さを軽減するロールベースの可観測性ツールを提供します。

以下は、クラスターで利用できる最もよく使用される Red Hat OpenShift Networking 機能の一部です。

- 次の Container Network Interface (CNI) プラグインのいずれかによって提供されるプライマリークラスターネットワーク:
  - OVN-Kubernetes ネットワークプラグイン - デフォルトのプラグイン
  - OpenShift SDN ネットワークプラグイン - OpenShift 4.14 以降のクラスターでは非推奨
- ネットワークプラグイン管理用の Cluster Network Operator

OpenShift 4.11 以降で作成された OpenShift Dedicated クラスターは、デフォルトで OVN-Kubernetes ネットワークプラグインを使用します。OpenShift バージョン 4.11 より前に作成された OpenShift Dedicated クラスターは、OpenShift バージョン 4.11 以降にアップグレードされた後も、OpenShift SDN プラグインを使用します。



重要

OpenShift Dedicated は、OpenShift Core Platform に基づく SDN のライフサイクルに従います。

- OpenShift バージョン 4.14 以降のクラスターでは SDN は非推奨です。
- OpenShift SDN プラグインをすでに使用しているクラスターは、OpenShift バージョン 4.11 以降にアップグレードした後も、引き続き SDN プラグインを使用します。
- クラスターは OpenShift バージョン 4.16 までアップグレードできます。
- OpenShift 4.16 で実行されているクラスター:
  - OpenShift バージョン 4.16 を使用しているクラスターは、SDN プラグインを使用している場合、アップグレードできません。
- SDN プラグインは OpenShift バージョン 4.17 で廃止されます。

OpenShift バージョン 4.15 以降で実行されているクラスターでは、まもなく OpenShift SDN から OVN に移行できるようになります。移行ツールはまだ提供されていません。OpenShift SDN の非推奨化と OVN 移行の詳細は、OCP 4.17 での OpenShift SDN CNI の削除 に関する KCS 記事を参照してください。

第2章 OPENSHIFT DEDICATED の DNS OPERATOR

OpenShift Dedicated の DNS Operator は、CoreDNS インスタンスをデプロイおよび管理して、クラスター内の Pod に名前解決サービスを提供し、DNS ベースの Kubernetes Service 検出を有効にし、内部の **cluster.local** 名を解決します。

2.1. DNS OPERATOR のステータスの確認

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、デーモンセットを使用して CoreDNS をデプロイし、デーモンセットのサービスを作成し、kubelet を Pod に対して名前解決に CoreDNS サービス 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 **Deployment** オブジェクトを使用してデプロイされます。

1. **oc get** コマンドを使用してデプロイメントのステータスを表示します。

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

出力例

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

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

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

出力例

```
NAME  VERSION     AVAILABLE  PROGRESSING  DEGRADED  SINCE  MESSAGE
dns   4.1.15-0.11  True       False        False     92m
```

AVAILABLE、**PROGRESSING**、および **DEGRADED** は、Operator のステータスに関する情報を示します。**AVAILABLE** が **True** になるのは、CoreDNS デーモンセットの 1 つ以上の Pod が **AVAILABLE** ステータス条件を報告し、DNS サービスがクラスター IP アドレスを持っている場合です。

2.2. 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
```

出力例

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

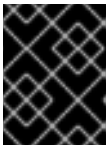
(1) Cluster Domain フィールドは、完全修飾 Pod およびサービスドメイン名を作成するために使用されるベース DNS ドメインです。

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

2.3. DNS 転送の使用

次の方法で、DNS 転送を使用して **/etc/resolv.conf** ファイル内のデフォルトの転送設定をオーバーライドできます。

- すべてのゾーンにネームサーバー (**spec.servers**) を指定します。転送されるゾーンが OpenShift Dedicated によって管理される Ingress ドメインである場合、アップストリームネームサーバーがドメインについて認証される必要があります。



重要

少なくとも 1 つのゾーンを指定する必要があります。そうしないと、クラスターの機能が失われる可能性があります。

- アップストリーム DNS サーバーのリスト (**spec.upstreamResolvers**) を指定します。
- デフォルトの転送ポリシーを変更します。

注記

デフォルトドメインの DNS 転送設定には、**/etc/resolv.conf** ファイルおよびアップストリーム DNS サーバーで指定されたデフォルトのサーバーの両方を設定できます。

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
```

上記のコマンドを実行すると、Operator が、**spec.servers** 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 config map を作成および更新します。

重要

zones パラメーターの値を指定する場合は、イントラネットなどの特定のゾーンにのみ転送してください。少なくとも 1 つのゾーンを指定する必要があります。そうしないと、クラスターの機能が失われる可能性があります。

クエリーに一致するゾーンがサーバーにない場合には、名前解決はアップストリーム DNS サーバーにフォールバックします。

DNS 転送の設定

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    negativeTTL: 0s
    positiveTTL: 0s
  logLevel: Normal
  nodePlacement: {}
  operatorLogLevel: Normal
  servers:
  - name: example-server (1)
    zones:
    - example.com (2)
    forwardPlugin:
      policy: Random (3)
      upstreams: (4)
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers: (5)
    policy: Random (6)
    protocolStrategy: "" (7)
    transportConfig: {} (8)
    upstreams:
    - type: SystemResolvConf (9)
    - type: Network
      address: 1.2.3.4 (10)
      port: 53 (11)
    status:
      clusterDomain: cluster.local
      clusterIP: x.y.z.10
      conditions:
    ...
```

(1) **rfc6335** サービス名の構文に準拠する必要があります。

(2) **rfc1123** サービス名構文のサブドメイン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** フィールドの無効なサブドメインです。

(3) **forwardPlugin** にリストされているアップストリームリゾルバーを選択するポリシーを定義します。デフォルト値は**Random**です。**RoundRobin** および **Sequential** の値を使用することもできます。

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

5 **upstreamResolvers** を使用すると、デフォルトの転送ポリシーをオーバーライドし、デフォルトドメインの指定された DNS リゾルバー (アップストリームリゾルバー) に DNS 解決を転送できます。アップストリームリゾルバーを提供しなかった場合、DNS 名のクエリーが **/etc/resolv.conf** で宣言されたサーバーに送信されます。

6 **upstreams** にリストされているアップストリームサーバーをクエリーのために選択する順序を決定します。**Random**、**RoundRobin**、または **Sequential** のいずれかの値を指定できます。デフォルト値は **Sequential** です。

7 省略すると、デフォルト (通常は元のクライアント要求のプロトコル) が選択されます。**TCP** に設定すると、クライアント要求が UDP を使用する場合でも、すべてのアップストリーム DNS 要求に対して TCP が必ず使用されます。

8 DNS 要求をアップストリームリゾルバーに転送するときに使用するトランスポートタイプ、サーバー名、およびオプションのカスタム CA または CA バンドルを設定するために使用されます。

9 2 種類の **upstreams** (**SystemResolvConf** または **Network**) を指定できます。**SystemResolvConf** で、アップストリームが **/etc/resolv.conf** を使用するように設定して、**Network** で **Networkresolver** を定義します。1 つまたは両方を指定できます。

10 指定したタイプが**Network**の場合には、IP アドレスを指定する必要があります。**address** フィールドは、有効な IPv4 または IPv6 アドレスである必要があります。

11 指定したタイプが **Network** の場合、必要に応じてポートを指定できます。**port** フィールドには **1** - **65535** の値を指定する必要があります。アップストリームのポートを指定しない場合、デフォルトのポートは 853 です。

関連情報

- 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。

2.4. DNS OPERATOR のステータスの確認

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
```

メッセージとスペルはリリースによって異なる場合がありますが、期待されるステータス出力は次のようになります。

```
Status:
  Conditions:
    Last Transition Time:  <date>
    Message:               DNS "default" is available.
    Reason:                AsExpected
    Status:                True
```

```
Type:                 Available
Last Transition Time:  <date>
Message:               Desired and current number of DNSes are equal
Reason:               AsExpected
Status:               False
Type:                 Progressing
Last Transition Time:  <date>
Reason:               DNSNotDegraded
Status:               False
Type:                 Degraded
Last Transition Time:  <date>
Message:               DNS default is upgradeable: DNS Operator can be upgraded
Reason:               DNSUpgradeable
Status:               True
Type:                 Upgradeable
```

2.5. DNS OPERATOR のログの表示

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

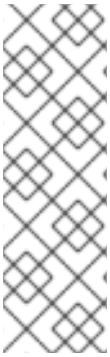
手順

- DNS Operator のログを表示します。

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

2.6. COREDNS ログレベルの設定

CoreDNS と CoreDNS Operator のログレベルは、それぞれ異なる方法を使用して設定します。CoreDNS ログレベルを設定して、ログに記録されたエラーメッセージの情報量を決定できます。CoreDNS ログレベルの有効な値は、**Normal**、**Debug**、および **Trace** です。デフォルトの **logLevel** は **Normal** です。



注記

CoreDNS のエラーログレベルは常に有効です。次のログレベル設定では、それぞれ異なるエラー応答が報告されます。

- **logLevel**: **Normal**は "errors" class: **log . { class error }** を有効にします。
- **logLevel**: **Debug**は "denial" class: **log . { class denial error }** を有効にします。
- **logLevel**: **Trace**は "all" class: **log . { class all }** を有効にします。

手順

- **logLevel** を **Debug** に設定するには、次のコマンドを入力します。

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --type=merge
```

- **logLevel** を **Trace** に設定するには、次のコマンドを入力します。

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --type=merge
```

検証

- 目的のログレベルが設定されていることを確認するには、config map を確認します。

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

たとえば、**logLevel** を **Trace** に設定すると、各サーバーブロックに次のスタンザが表示されます。

```
errors
log . {
    class all
}
```

2.7. COREDNS OPERATOR のログレベルの設定

CoreDNS と CoreDNS Operator のログレベルは、それぞれ異なる方法を使用して設定します。クラスター管理者は、Operator ログレベルを設定して、OpenShift DNS の問題をより迅速に追跡できます。**operatorLogLevel** の有効な値は、**Normal**、**Debug**、および**Trace**です。**Trace** には最も詳細にわたる情報が含まれます。デフォルトの**operatorlogLevel**は**Normal**です。Operator の問題のログレベルには、Trace、Debug、Info、Warning、Error、Fatal、および Panic の 7 つがあります。ログレベルの設定後に、その重大度またはそれを超える重大度のログエントリーがログに記録されます。

- **operatorLogLevel: "Normal"** は **logrus.SetLogLevel("Info")** を設定します。
- **operatorLogLevel: "Debug"** は **logrus.SetLogLevel("Debug")** を設定します。
- **operatorLogLevel: "Trace"** は **logrus.SetLogLevel("Trace")** を設定します。

手順

- **operatorLogLevel**を**Debug**に設定するには、次のコマンドを入力します。

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Debug"}}' --
type=merge
```

- **operatorLogLevel**を**Trace**に設定するには、次のコマンドを入力します。

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Trace"}}' --
type=merge
```

検証

1. 結果の変更を確認するには、次のコマンドを入力します。

```
$ oc get dnses.operator -A -oyaml
```

2 つのログレベルのエントリーが表示されるはずです。**operatorLogLevel** は OpenShift DNS Operator の問題に適用され、**logLevel** は CoreDNS Pod のデーモンセットに適用されます。

```
logLevel: Trace
operatorLogLevel: Debug
```

2. デーモンセットのログを確認するには、次のコマンドを入力します。

```
$ oc logs -n openshift-dns ds/dns-default
```

2.8. COREDNS キャッシュのチューニング

CoreDNS の場合、成功または失敗したキャッシュ (それぞれポジティブキャッシュまたはネガティブキャッシュとも呼ばれます) の最大期間を設定できます。DNS クエリー応答のキャッシュ期間をチューニングすると、アップストリーム DNS リゾルバーの負荷を軽減できます。

警告

TTL フィールドを低い値に設定すると、クラスター、上流のリゾルバー、またはその両方の負荷が増加する可能性があります。

手順

1. 次のコマンドを実行して、**default** という名前の DNS Operator オブジェクトを編集します。

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

2. Time-to-Live (TTL) キャッシュ値を変更します。

DNS キャッシングの設定

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h (1)
    negativeTTL: 0.5h10m (2)
```

(1) 文字列値 **1h** は、CoreDNS によってそれぞれの秒数に変換されます。このフィールドを省略した場合、値は **0** と見なされ、クラスターはフォールバックとして内部デフォルト値の **900** を使用します。

(2) 文字列値は、**0.5h10m** などの単位の組み合わせにすることができ、CoreDNS によってそれぞれの秒数に変換されます。このフィールドを省略した場合、値は **0** と見なされ、クラスターはフォールバックとして内部デフォルト値の **30** を使用します。

検証

1. 変更を確認するには、次のコマンドを実行して config map を再度確認します。

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

2. 次の例のようなエントリーが表示されていることを確認します。

```
cache 3600 {
    denial 9984 2400
}
```

関連情報

キャッシュの詳細は、CoreDNS cache を参照してください。

2.9. 高度なタスク

2.9.1. DNS Operator managementState の変更

DNS Operator は、CoreDNS コンポーネントを管理し、クラスター内の Pod とサービスに名前解決サービスを提供します。DNS Operator の **managementState** は、デフォルトで **Managed** に設定されます。これは、DNS Operator がそのリソースをアクティブに管理していることを意味します。これを **Unmanaged** に変更できます。つまり、DNS Operator がそのリソースを管理していないことを意味します。

以下は、DNS Operator **managementState** を変更するためのユースケースです。

- 開発者は、CoreDNS の問題が修正されているかどうかを確認するために、設定変更をテストする必要があります。**managementState** を **Unmanaged** に設定することで、DNS Operator による設定変更の上書きを防止できます。
- クラスター管理者は、CoreDNS の問題を報告していますが、問題が修正されるまで回避策を適用する必要があります。DNS Operator の **managementState** フィールドを **Unmanaged** に設定して、回避策を適用できます。

手順

1. DNS Operator の **managementState** を **Unmanated** に変更します。

```
oc patch dns.operator.openshift.io default --type merge --patch '{"spec":
{"managementState":"Unmanaged"}}'
```

2. **jsonpath** コマンドライン JSON パーサーを使用して DNS Operator の **managementState** を確認します。

```
$ oc get dns.operator.openshift.io default -ojsonpath='{.spec.managementState}'
```

出力例

```
"Unmanaged"
```

注記

managementState が **Unmanaged** に設定されている間はアップグレードできません。

2.9.2. DNS Pod 配置の制御

DNS Operator には 2 つのデーモンセットがあります。1 つは CoreDNS 用の **dns-default** という名前のデーモンセットで、もう 1 つは **/etc/hosts** ファイルの管理用の **node-resolver** という名前のデーモンセットです。

場合によっては、どのノードに CoreDNS Pod を割り当てて実行するかを制御する必要があります (ただし、これは一般的な操作ではありません)。たとえば、クラスター管理者がノードのペア間の通信を禁止できるセキュリティーポリシーを設定している場合、CoreDNS のデーモンセットが実行されるノードのセットを制限する必要があります。DNS Pod がクラスター内の一部のノードで実行されており、DNS Pod が実行されていないノードから DNS Pod が実行されているノードへのネットワーク接続がある場合、すべての Pod で DNS サービスが利用可能になります。

node-resolver デーモンセットは、すべてのノードホストで実行する必要があります。このデーモンセットにより、イメージのプルをサポートするクラスターイメージレジストリーのエントリーが追加されるためです。**node-resolver** Pod には、1 つのジョブのみがあります。コンテナーランタイムがサービス名を解決できるように、**image-registry.openshift-image-registry.svc** サービスのクラスター IP アドレスを検索し、それをノードホストの **/etc/hosts** に追加するジョブです。

クラスター管理者は、カスタムノードセレクターを使用して、CoreDNS のデーモンセットを特定のノードで実行するか、実行しないように設定できます。

前提条件

- **oc** CLI がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。
- DNS Operator の **managementState** が **Managed** に設定されている。

手順

- CoreDNS のデーモンセットを特定のノードで実行できるようにするために、taint と toleration を設定します。
  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
     ```

  2. taint の taint キーおよび toleration を指定します。

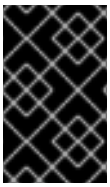
     ```
     spec:
       nodePlacement:
         tolerations:
         - effect: NoExecute
           key: "dns-only"
           operators: Equal
           value: abc
           tolerationSeconds: 3600 1
     ```

     1 taint が **dns-only** である場合、それは無制限に許容できます。**tolerationSeconds** は省略できます。

2.9.3. TLS を使用した DNS 転送の設定

高度に規制された環境で作業する場合は、要求をアップストリームリゾルバーに転送する際に DNS トラフィックのセキュリティーを確保して、追加の DNS トラフィックおよびデータのプライバシーを確保できるようにする必要がある場合があります。

CoreDNS は転送された接続を 10 秒間キャッシュすることに注意してください。要求が発行されない場合、CoreDNS はその 10 秒間、TCP 接続を開いたままにします。大規模なクラスターでは、ノードごとに接続を開始できるため、DNS サーバーが多くの新しい接続を開いたまま保持する可能性があることを認識しているか確認してください。パフォーマンスの問題を回避するために、それに応じて DNS 階層を設定します。



重要

zones パラメーターの値を指定する場合は、イントラネットなどの特定のゾーンにのみ転送してください。少なくとも 1 つのゾーンを指定する必要があります。そうしないと、クラスターの機能が失われる可能性があります。

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
```

クラスター管理者は、転送された DNS クエリーに Transport Layer Security (TLS) を設定できるようになりました。

TLS を使用した DNS 転送の設定

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: example-server 1
    zones:
    - example.com 2
    forwardPlugin:
      transportConfig:
        transport: TLS 3
        tls:
          caBundle:
            name: mycacert
          serverName: dnstls.example.com 4
      policy: Random 5
      upstreams: 6
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers: 7
    transportConfig:
      transport: TLS
      tls:
        caBundle:
          name: mycacert
        serverName: dnstls.example.com
```

```
  upstreams:
  - type: Network 8
    address: 1.2.3.4 9
    port: 53 10
```

1. **rfc6335** サービス名の構文に準拠する必要があります。
2. **rfc1123** サービス名構文のサブドメイン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** フィールドの無効なサブドメインです。クラスタードメインの **cluster.local** は、**zones** の無効な **subdomain** です。
3. 転送された DNS クエリーの TLS を設定する場合、**transport** フィールドの値を **TLS** に設定します。
4. 転送された DNS クエリー用に TLS を設定する場合、これは、アップストリーム TLS サーバー証明書を検証するための Server Name Indication (SNI) の一部として使用される必須のサーバー名です。
5. アップストリームリゾルバーを選択するためのポリシーを定義します。デフォルト値は**Random**です。**RoundRobin** および **Sequential** の値を使用することもできます。
6. 必須。アップストリームリゾルバーを指定するために使用します。**forwardPlugin** エントリーごとに最大 15 の **upstreams** エントリーが許可されます。
7. オプション: これを使用して、デフォルトポリシーを上書きし、デフォルトドメインで指定された DNS リゾルバー (アップストリームリゾルバー) に DNS 解決を転送できます。アップストリームリゾルバーを指定しない場合に、DNS 名のクエリーは **/etc/resolv.conf** のサーバーに送信されます。
8. TLS を使用する場合、**Network** タイプのみが許可され、IP アドレスを指定する必要があります。**Network** タイプは、このアップストリームリゾルバーが **/etc/resolv.conf** にリストされているアップストリームリゾルバーとは別に転送されたリクエストを処理する必要があることを示します。
9. **address** フィールドは、有効な IPv4 または IPv6 アドレスである必要があります。
10. オプションでポートを指定できます。**port** の値は **1** ～ **65535** である必要があります。アップストリームのポートを指定しない場合、デフォルトのポートは 853 です。

注記

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

検証

1. config map を表示します。

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

TLS 転送の例に基づく DNS ConfigMap のサンプル

```
apiVersion: v1
data:
```

```
  Corefile: |
    example.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 1.2.3.4:53 {
            policy Random
        }
        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 デーモンセットのローリング更新がトリガーされます。

関連情報

- 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3章 OPENSHIFT DEDICATED の INGRESS OPERATOR

3.1. OPENSHIFT DEDICATED の INGRESS OPERATOR

OpenShift Dedicated クラスターの作成時に、クラスターで実行される Pod およびサービスにはそれぞれ独自の IP アドレスが割り当てられます。IP アドレスは、近くで実行されている他の Pod やサービスからアクセスできますが、外部クライアントの外部からはアクセスできません。Ingress Operator は **IngressController** API を実装し、OpenShift Dedicated 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。

Ingress Operator を使用すると、ルーティングを処理する 1 つ以上の HAProxy ベースの Ingress Controller をデプロイおよび管理することにより、外部クライアントがサービスにアクセスできるようになります。Red Hat の Site Reliability Engineers (SRE) は、OpenShift Dedicated クラスターの Ingress Operator を管理します。Ingress Operator の設定を変更することはできませんが、デフォルトの Ingress コントローラーの設定、ステータス、およびログおよび Ingress Operator ステータスを表示できます。

3.2. 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 Controller のドメインとして使用します。
- OpenShift API Server Operator は、クラスター Ingress 設定からのドメインを使用します。このドメインは、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も使用されます。

3.3. INGRESS CONTROLLER 設定パラメーター

IngressController カスタムリソース (CR) には、組織の特定のニーズを満たすように設定できる任意の設定パラメーターが含まれています。

パラメーター	説明

パラメーター	説明
domain	**domain** は Ingress Controller 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 Controller 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 Controller レプリカの数です。設定されていない場合、デフォルト値は **2** になります。

パラメーター	説明
endpointPublishingStrategy	**endpointPublishingStrategy** は Ingress Controller 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。 クラウド環境の場合、**loadBalancer** フィールドを使用して、Ingress Controller のエンドポイント公開ストラテジーを設定します。 次の **endpointPublishingStrategy** フィールドを設定できます。 ● **loadBalancer.scope** ● **loadBalancer.allowedSourceRanges** 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。 ● Amazon Web Services (AWS): **LoadBalancerService** (外部スコープあり) ● Google Cloud Platform (GCP): **LoadBalancerService** (外部スコープあり) ほとんどのプラットフォームでは、**endpointPublishingStrategy** 値は更新できます。GCP では、次の**endpointPublishingStrategy**フィールドを設定できます。 ● **loadBalancer.scope** ● **loadbalancer.providerParameters.gcp.clientAccess** クラスターのデプロイ後に、**endpointPublishingStrategy** の値を更新する必要がある場合は、次の **endpointPublishingStrategy** フィールドを設定できます。 ● **hostNetwork.protocol** ● **nodePort.protocol** ● **private.protocol**
defaultCertificate	**defaultCertificate** 値は、Ingress Controller 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への参照です。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。 使用中の証明書は、生成されたものでもユーザーが指定したものでも、OpenShift Dedicated のビルトイン OAuth サーバーと自動的に統合されます。

パラメーター	説明
namespaceSelector	**namespaceSelector** は、Ingress Controller によって提供される 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 Controller 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 Controller 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。以下に例を示します。 nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists

パラメーター	説明
tlsSecurityProfile	**tlsSecurityProfile** は、Ingress Controller の TLS 接続の設定を指定します。 これが設定されていない場合、デフォルト値は **apiservers.config.openshift.io/cluster** リソースをベースとして設定されます。 **Old**、**Intermediate**、および **Modern** のプロファイルタイプを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース **X.Y.Z** にデプロイされた **Intermediate** プロファイルを使用する仕様がある場合、リリース **X.Y.Z+1** へのアップグレードにより、新規のプロファイル設定が Ingress Controller に適用され、ロールアウトが生じる可能性があります。 Ingress Controller の最小 TLS バージョンは **1.1** で、最大 TLS バージョンは**1.3** です。  **注記** 設定されたセキュリティープロファイルの暗号および最小 TLS バージョンが **TLSProfile** ステータスに反映されます。  **重要** Ingress Operator は TLS **1.0** の **Old** または **Custom** プロファイルを **1.1** に変換します。
clientTLS	**clientTLS** は、クラスターおよびサービスへのクライアントアクセスを認証します。その結果、相互 TLS 認証が有効になります。設定されていない場合、クライアント TLS は有効になっていません。 **clientTLS** には、必要なサブフィールド **spec.clientTLS.clientCertificatePolicy** および **spec.clientTLS.ClientCA** があります。 **ClientCertificatePolicy** サブフィールドは、**Required** または **Optional** の 2 つの値のいずれかを受け入れます。**ClientCA** サブフィールドは、openshift-config namespace にある config map を指定します。config map には CA 証明書バンドルが含まれている必要があります。 **AllowedSubjectPatterns** は、要求をフィルターするために有効なクライアント証明書の識別名と照合される正規表現のリストを指定する任意の値です。正規表現は PCRE 構文を使用する必要があります。1 つ以上のパターンがクライアント証明書の識別名と一致している必要があります。一致しない場合、Ingress Controller は証明書を拒否し、接続を拒否します。指定しないと、Ingress Controller は識別名に基づいて証明書を拒否しません。

パラメーター	説明
routeAdmission	**routeAdmission** は、複数の namespace での要求の許可または拒否など、新規ルート要求を処理するためのポリシーを定義します。 **namespaceOwnership** は、namespace 間でホスト名の要求を処理する方法を記述します。デフォルトは **Strict** です。 ● **Strict**: ルートが複数の namespace 間で同じホスト名を要求することを許可しません。 ● **InterNamespaceAllowed**: ルートが複数の namespace 間で同じホスト名の異なるパスを要求することを許可します。 **wildcardPolicy** は、ワイルドカードポリシーを使用するルートが Ingress Controller によって処理される方法を記述します。 ● **WildcardsAllowed**: ワイルドカードポリシーと共にルートが Ingress Controller によって許可されていることを示します。 ● **WildcardsDisallowed**: ワイルドカードポリシーの **None** を持つルートのみが Ingress Controller によって許可されることを示します。**wildcardPolicy** を **WildcardsAllowed** から **WildcardsDisallowed** に更新すると、ワイルドカードポリシーの **Subdomain** を持つ許可されたルートが機能を停止します。これらのルートは、Ingress Controller によって許可されるように **None** のワイルドカードポリシーに対して再作成される必要があります。**WildcardsDisallowed** はデフォルト設定です。

パラメーター	説明
IngressControllerLogging	**logging** はログに記録される内容および場所のパラメーターを定義します。このフィールドが空の場合、操作ログは有効になりますが、アクセスログは無効になります。 ● **access** は、クライアント要求をログに記録する方法を記述します。このフィールドが空の場合、アクセスロギングは無効になります。 ○ **destination** はログメッセージの宛先を記述します。 ■ **type** はログの宛先のタイプです。 ● **Container** は、ログがサイドカーコンテナーに移動することを指定します。Ingress Operator は Ingress Controller Pod で **logs** という名前のコンテナーを設定し、Ingress Controller がログをコンテナーに書き込むように設定します。管理者がこのコンテナーからログを読み取るカスタムロギングソリューションを設定することが予想されます。コンテナーログを使用すると、ログの割合がコンテナーランタイムの容量やカスタムロギングソリューションの容量を超えるとログがドロップされることがあります。 ● **Syslog** は、ログが Syslog エンドポイントに送信されることを指定します。管理者は、Syslog メッセージを受信できるエンドポイントを指定する必要があります。管理者がカスタム Syslog インスタンスを設定していることが予想されます。 ■ **container** は **Container** ロギング宛先タイプのパラメーターを記述します。現在、コンテナーロギングのパラメーターはないため、このフィールドは空である必要があります。 ■ **syslog** は、**Syslog** ロギング宛先タイプのパラメーターを記述します。 ● **address** は、ログメッセージを受信する syslog エンドポイントの IP アドレスです。 ● **port** は、ログメッセージを受信する syslog エンドポイントの UDP ポート番号です。 ● **maxLength**は、syslog メッセージの最大長です。サイズは **480** から **4096** バイトである必要があります。このフィールドが空の場合には、最大長はデフォルト値の**1024**バイトに設定されます。 ● **facility** はログメッセージの syslog ファシリティーを指定します。このフィールドが空の場合、ファシリティーは **local1** になります。それ以外の場合、有効な syslog ファシリティー (**kern**、**user**、**mail**、**daemon**、**auth**、**syslog**、**lpr**、**news**、**uucp**、**cron**、**auth2**、**ftp**、**ntp**、**audit**、**alert**、**cron2**、**local0**、**local1**、**local2**、**local3**) を指定する必要があります。**local4**、**local5**、**local6**、または **local7**。 ○ **httpLogFormat** は、HTTP 要求のログメッセージの形式を指定します。このフィールドが空の場合、ログメッセージは実装のデフォルト HTTP ログ形式を使用します。HAProxy のデフォルトの HTTP ログ形式については、HAProxy ドキュメント を参照してください。

パラメーター	説明
httpHeaders	**httpHeaders** は HTTP ヘッダーのポリシーを定義します。
	IngressControllerHTTPHeaders の **forwardedHeaderPolicy** を設定することで、Ingress Controller が **Forwarded**、**X-Forwarded-For**、**X-Forwarded-Host**、**X-Forwarded-Port**、**X-Forwarded-Proto**、および **X-Forwarded-Proto-Version** HTTP ヘッダーをいつどのように設定するか指定します。 デフォルトでは、ポリシーは **Append** に設定されます。 ● **Append** は、Ingress Controller がヘッダーを追加するように指定し、既存のヘッダーを保持します。 ● **Replace** は、Ingress Controller がヘッダーを設定するように指定し、既存のヘッダーを削除します。 ● **IfNone** は、ヘッダーがまだ設定されていない場合に、Ingress Controller がヘッダーを設定するように指定します。 ● **Never** は、Ingress Controller がヘッダーを設定しないように指定し、既存のヘッダーを保持します。 **headerNameCaseAdjustments** を設定して、HTTP ヘッダー名に適用できるケースの調整を指定できます。それぞれの調整は、必要な大文字化を指定して HTTP ヘッダー名として指定されます。たとえば、**X-Forwarded-For** を指定すると、指定された大文字化を有効にするために **x-forwarded-for** HTTP ヘッダーを調整する必要があることを示唆できます。 これらの調整は、クリアテキスト、edge-terminated、および re-encrypt ルートにのみ適用され、HTTP/1 を使用する場合にのみ適用されます。 要求ヘッダーの場合、これらの調整は **haproxy.router.openshift.io/h1-adjust-case=true** アノテーションを持つルートにのみ適用されます。応答ヘッダーの場合、これらの調整はすべての HTTP 応答に適用されます。このフィールドが空の場合、要求ヘッダーは調整されません。 **actions** は、ヘッダーに対して特定のアクションを実行するためのオプションを指定します。TLS パススルー接続のヘッダーは設定または削除できません。**actions** フィールドには、**spec.httpHeader.actions.response** および **spec.httpHeader.actions.request** の追加のサブフィールドがあります。 ● **response** サブフィールドは、設定または削除する HTTP 応答ヘッダーのリストを指定します。 ● **request** サブフィールドは、設定または削除する HTTP 要求ヘッダーのリストを指定します。
httpCompression	**httpCompression**は、HTTP トラフィック圧縮のポリシーを定義します。 ● **mimeTypes** は、圧縮を適用する必要がある MIME タイプのリストを定義します。(例: **text/css; charset=utf-8**, **text/html**, **text/***, **image/svg+xml**, **application/octet-stream**, **X-custom/customsub**, using the format pattern,**type/subtype; [;attribute=value]**)**types** は、アプリケーション、イメージ、メッセージ、マルチパート、テキスト、ビデオ、または **X-** で始まるカスタムタイプ。例: MIME タイプとサブタイプの完全な表記を確認するには、RFC1341 を参照してください。

パラメーター	説明
httpErrorCodePages	**httpErrorCodePages** は、カスタムの HTTP エラーコードの応答ページを指定します。デフォルトで、IngressController は IngressController イメージにビルドされたエラーページを使用します。

httpCaptureCookies	**httpCaptureCookies** は、アクセスログにキャプチャーする HTTP Cookie を指定します。**httpCaptureCookies** フィールドが空の場合、アクセスログは Cookie をキャプチャーしません。
	
	キャプチャーするすべての Cookie について、次のパラメーターが **IngressController** 設定に含まれている必要があります。
	
	● **name** は、Cookie の名前を指定します。
	● **maxLength** は、Cookie の最大長を指定します。
	● **matchType** は、Cookie のフィールドの **name** が、キャプチャー Cookie 設定と完全に一致するか、キャプチャー Cookie 設定の接頭辞であるかを指定します。**matchType** フィールドは **Exact** および **Prefix** パラメーターを使用します。
	
	以下に例を示します。
	
	`httpCaptureCookies:`
	`- matchType: Exact`
	`  maxLength: 128`
	`  name: MYCOOKIE`

パラメーター	説明

httpCaptureHeaders	**httpCaptureHeaders** は、アクセスログにキャプチャーする HTTP ヘッダーを指定します。**httpCaptureHeaders** フィールドが空の場合、アクセスログはヘッダーをキャプチャーしません。
	
	httpCaptureHeaders には、アクセスログにキャプチャーするヘッダーの 2 つのリストが含まれています。ヘッダーフィールドの 2 つのリストは **request** と **response** です。どちらのリストでも、**name** フィールドはヘッダー名を指定し、**maxlength** フィールドはヘッダーの最大長を指定する必要があります。以下に例を示します。
	
	`httpCaptureHeaders:`
	`  request:`
	`  - maxLength: 256`
	`    name: Connection`
	`  - maxLength: 128`
	`    name: User-Agent`
	`  response:`
	`  - maxLength: 256`
	`    name: Content-Type`
	`  - maxLength: 256`
	`    name: Content-Length`

tuningOptions	**tuningOptions** は、Ingress Controller Pod のパフォーマンスを調整するためのオプションを指定します。 ● **clientFinTimeout** は、クライアントの応答が接続を閉じるのを待機している間に接続が開かれる期間を指定します。デフォルトのタイムアウトは **1s** です。 ● **clientTimeout** は、クライアント応答の待機中に接続が開かれる期間を指定します。デフォルトのタイムアウトは **30s** です。 ● **headerBufferBytes** は、Ingress Controller 接続セッション用に予約されるメモリーの量をバイト単位で指定します。Ingress Controller で HTTP / 2 が有効になっている場合、この値は少なくとも **16384** である必要があります。設定されていない場合、デフォルト値は **32768** バイトになります。このフィールドを設定することは推奨しません。**headerBufferBytes** 値が小さすぎると Ingress Controller が破損する可能性があり、**headerBufferBytes** 値が大きすぎると、Ingress Controller が必要以上のメモリーを使用する可能性があるためです。 ● **headerBufferMaxRewriteBytes** は、HTTP ヘッダーの書き換えと Ingress Controller 接続セッションの追加のために **headerBufferBytes** から予約するメモリーの量をバイト単位で指定します。**headerBufferMaxRewriteBytes** の最小値は **4096** です。受信 HTTP 要求には、**headerBufferBytes** は **headerBufferMaxRewriteBytes** よりも大きくなければなりません。設定されていない場合、デフォルト値は **8192** バイトになります。このフィールドを設定することは推奨しません。**headerBufferMaxRewriteBytes** 値が小さすぎると Ingress Controller が破損する可能性があり、**headerBufferMaxRewriteBytes** 値が大きすぎると、Ingress Controller が必要以上のメモリーを使用する可能性があるためです。 ● **healthCheckInterval** は、ルーターがヘルスチェックの間隔として待機する時間を指定します。デフォルトは **5s** です。

パラメーター	説明
	● **serverFinTimeout** は、接続を閉じるクライアントへの応答を待つ間、接続が開かれる期間を指定します。デフォルトのタイムアウトは **1s** です。 ● **serverTimeout** は、サーバーの応答を待機している間に接続が開かれる期間を指定します。デフォルトのタイムアウトは **30s** です。 ● **threadCount** は、HAProxy プロセスごとに作成するスレッドの数を指定します。より多くのスレッドを作成すると、使用されるシステムリソースを増やすことで、各 Ingress Controller Pod がより多くの接続を処理できるようになります。HAProxy は最大 **64** のスレッドをサポートします。このフィールドが空の場合、Ingress Controller はデフォルト値の **4** スレッドを使用します。デフォルト値は、将来のリリースで変更される可能性があります。このフィールドを設定することは推奨しません。HAProxy スレッドの数を増やすと、Ingress Controller Pod が負荷時に CPU 時間をより多く使用できるようになり、他の Pod が実行に必要な CPU リソースを受け取れないようになるためです。スレッドの数を減らすと、Ingress Controller のパフォーマンスが低下する可能性があります。 ● **tlsInspectDelay** は、一致するルートを見つけるためにルーターがデータを保持する期間を指定します。この値の設定が短すぎると、より一致する証明書を使用している場合でも、ルーターがエッジ終端、再暗号化された、またはパススルーのルートのデフォルトの証明書にフォールバックする可能性があります。デフォルトの検査遅延は **5s** です。 ● **tunnelTimeout** は、トンネルがアイドル状態の間、websocket などのトンネル接続期間を開いた期間を指定します。デフォルトのタイムアウトは **1h** です。 ● **maxConnections** は、HAProxy プロセスごとに確立できる同時接続の最大数を指定します。この値を増やすと、追加のシステムリソースで各 Ingress Controller Pod がより多くの接続を処理できるようになります。**0**、**-1**、**2000** から **2000000** の範囲内の任意の値を使用でき、フィールドを空にすることも可能です。 ○ このフィールドが空のままであるか、値が **0** の場合、Ingress Controller はデフォルト値の **50000** を使用します。この値は、今後のリリースで変更される可能性があります。 ○ フィールド値が **-1** の場合、HAProxy は、実行中のコンテナーで使用可能な **ulimit** に基づき最大値を動的に計算します。このプロセスにより、計算値が大きくなり、現在のデフォルト値である **50000** と比較してかなり大きなメモリー使用量が発生します。 ○ フィールドの値が現在のオペレーティングシステムの制限よりも大きい場合、HAProxy プロセスは開始されません。 ○ 個別の値を選択し、ルーター Pod が新しいノードに移行された場合、新しいノードに同一の **ulimit** が設定されていない可能性があります。このような場合、Pod は起動に失敗します。 ○ 異なる **ulimit** を持つノードが設定されていて、離散値を選択する場合は、実行時に接続の最大数が計算されるように、このフィールドに **-1** の値を使用することを推奨します。

パラメーター	説明
logEmptyRequests	**logEmptyRequests** は、リクエストを受け取らず、ログに記録されない接続を指定します。これらの空の要求は、ロードバランサーヘルスプローブまたは Web ブラウザーの投機的接続 (事前接続) から送信され、これらの要求をログに記録することは望ましくない場合があります。ただし、これらの要求はネットワークエラーによって引き起こされる可能性があります。この場合は、空の要求をログに記録すると、エラーの診断に役立ちます。これらの要求はポートスキャンによって引き起こされ、空の要求をログに記録すると、侵入の試行が検出されなくなります。このフィールドに使用できる値は **Log** および **Ignore** です。デフォルト値は **Log** です。 **LoggingPolicy** タイプは、以下のいずれかの値を受け入れます。 ● **ログ**: この値を **Log** に設定すると、イベントがログに記録される必要があることを示します。 ● **Ignore**: この値を **Ignore** に設定すると、HAproxy 設定の **dontlognull** オプションを設定します。
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy** は、リクエストを受け取る前に接続がタイムアウトした場合に HTTP 接続を処理する方法を記述します。このフィールドに使用できる値は **Respond** および **Ignore** です。デフォルト値は **Respond** です。 **HTTPEmptyRequestsPolicy** タイプは、以下のいずれかの値を受け入れます。 ● **応答**: フィールドが **Respond** に設定されている場合、Ingress Controller は HTTP **400** または **408** 応答を送信する場合、アクセスログが有効な場合に接続をログに記録し、適切なメトリックで接続をカウントします。 ● **ignore**: このオプションを **Ignore** に設定すると HAproxy 設定に **http-ignore-probes** パラメーターが追加されます。フィールドが **Ignore** に設定されている場合、Ingress Controller は応答を送信せずに接続を閉じると、接続をログに記録するか、メトリックを増分します。 これらの接続は、ロードバランサーのヘルスプローブまたは Web ブラウザーの投機的接続 (事前接続) から取得され、無視しても問題はありません。ただし、これらの要求はネットワークエラーによって引き起こされる可能性があります。そのため、このフィールドを **Ignore** に設定すると問題の検出と診断が妨げられる可能性があります。これらの要求はポートスキャンによって引き起こされ、空の要求をログに記録すると、侵入の試行が検出されなくなります。

3.3.1. Ingress Controller の TLS セキュリティープロファイル

TLS セキュリティープロファイルは、サーバーに接続する際に接続クライアントが使用できる暗号を規制する方法をサーバーに提供します。

3.3.1.1. TLS セキュリティープロファイルについて

TLS (Transport Layer Security) セキュリティープロファイルを使用すると、さまざまな OpenShift Dedicated コンポーネントで必須にする TLS 暗号を定義できます。OpenShift Dedicated の TLS セキュリティープロファイルは、Mozilla の推奨設定 に基づいています。

コンポーネントごとに、以下の TLS セキュリティープロファイルのいずれかを指定できます。

表3.1 TLS セキュリティープロファイル

プロファイル	説明
Old	このプロファイルは、レガシークライアントまたはライブラリーでの使用を目的としています。このプロファイルは、Old 後方互換性 の推奨設定に基づいています。 **Old** プロファイルには、最小 TLS バージョン 1.0 が必要です。 **注記** Ingress Controller の場合、TLS の最小バージョンは 1.0 から 1.1 に変換されます。
Intermediate	このプロファイルは、大多数のクライアントに推奨される設定です。これは、Ingress Controller、kubelet、およびコントロールプレーンのデフォルトの TLS セキュリティープロファイルです。このプロファイルは、Intermediate 互換性 の推奨設定に基づいています。 **Intermediate** プロファイルには、最小 TLS バージョン 1.2 が必要です。
Modern	このプロファイルは、後方互換性を必要としない Modern のクライアントでの使用を目的としています。このプロファイルは、Modern 互換性 の推奨設定に基づいています。 **Modern** プロファイルには、最小 TLS バージョン 1.3 が必要です。
カスタム	このプロファイルを使用すると、使用する TLS バージョンと暗号を定義できます。 **警告** 無効な設定により問題が発生する可能性があるため、**Custom** プロファイルを使用する際には注意してください。

注記

事前定義されたプロファイルタイプのいずれかを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース X.Y.Z にデプロイされた Intermediate プロファイルを使用する仕様がある場合、リリース X.Y.Z+1 へのアップグレードにより、新規のプロファイル設定が適用され、ロールアウトが生じる可能性があります。

3.3.1.2. Ingress Controller の TLS セキュリティープロファイルの設定

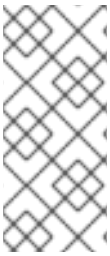
Ingress Controller の TLS セキュリティープロファイルを設定するには、**IngressController** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合、デフォルト値は API サーバーに設定された TLS セキュリティープロファイルに基づいています。

Old TLS のセキュリティープロファイルを設定するサンプル IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS セキュリティープロファイルは、Ingress Controller の TLS 接続の最小 TLS バージョンと TLS 暗号を定義します。

設定された TLS セキュリティープロファイルの暗号と最小 TLS バージョンは、**Status.Tls Profile** 配下の **IngressController** カスタムリソース (CR) と **Spec.Tls Security Profile** 配下の設定された TLS セキュリティープロファイルで確認できます。**Custom** TLS セキュリティープロファイルの場合、特定の暗号と最小 TLS バージョンは両方のパラメーターの下に一覧表示されます。



注記

HAProxy Ingress Controller イメージは、TLS**1.3** と **Modern** プロファイルをサポートしています。

また、Ingress Operator は TLS **1.0** の **Old** または **Custom** プロファイルを **1.1** に変換します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **openshift-ingress-operator** プロジェクトの **IngressController** CR を編集して、TLS セキュリティープロファイルを設定します。

   ```
   $ oc edit IngressController default -n openshift-ingress-operator
   ```

2. **spec.tlsSecurityProfile** フィールドを追加します。

Custom プロファイルのサンプル IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
...
```

(1) TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。

(2) 選択したタイプに適切なフィールドを指定します。

- **old: {}**
- **intermediate: {}**
- **custom:**

(3) **custom** タイプには、TLS 暗号のリストと最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

検証

- **IngressController** CR にプロファイルが設定されていることを確認します。

```
$ oc describe IngressController default -n openshift-ingress-operator
```

出力例

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
```

```
      ECDHE-RSA-CHACHA20-POLY1305
      ECDHE-RSA-AES128-GCM-SHA256
      ECDHE-ECDSA-AES128-GCM-SHA256
    Min TLS Version:  VersionTLS11
  Type:             Custom
...
```

3.3.1.3. 相互 TLS 認証の設定

spec.clientTLS 値を設定して、相互 TLS (mTLS) 認証を有効にするように Ingress Controller を設定できます。**clientTLS** 値は、クライアント証明書を検証するように Ingress Controller を設定します。この設定には、config map の参照である **clientCA** 値の設定が含まれます。config map には、クライアントの証明書を検証するために使用される PEM でエンコードされた CA 証明書バンドルが含まれます。必要に応じて、証明書サブジェクトフィルターのリストも設定できます。

clientCA 値が X509v3 証明書失効リスト (CRL) ディストリビューションポイントを指定している場合、Ingress Operator は、提供された各証明書で指定されている HTTP URI X509v3 **CRL Distribution Point** に基づいて CRL config map をダウンロードおよび管理します。Ingress Controller は、mTLS/TLS ネゴシエーション中にこの config map を使用します。有効な証明書を提供しない要求は拒否されます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- PEM でエンコードされた CA 証明書バンドルがある。
- CA バンドルが CRL ディストリビューションポイントを参照する場合は、エンドエンティティーまたはリーフ証明書もクライアント CA バンドルに含める必要があります。この証明書には、RFC 5280 で説明されているとおり、この証明書の **CRL Distribution Points** に HTTP URI が含まれている必要があります。以下に例を示します。

```
Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
    Subject: SOME SIGNED CERT          X509v3 CRL Distribution Points:
        Full Name:
          URI:http://crl.example.com/example.crl
```

手順

1. **openshift-config** namespace で、CA バンドルから config map を作成します。

```
$ oc create configmap \
  router-ca-certs-default \
  --from-file=ca-bundle.pem=client-ca.crt \ 1
  -n openshift-config
```

1 config map データキーは **ca-bundle.pem** で、data の値は PEM 形式の CA 証明書である必要があります。

2. **openshift-ingress-operator** プロジェクトで **IngressController** リソースを編集します。

```
$ oc edit IngressController default -n openshift-ingress-operator
```

3. **spec.clientTLS** フィールドおよびサブフィールドを追加して相互 TLS を設定します。

フィルタリングパターンを指定する clientTLS プロファイルのサンプル IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
    allowedSubjectPatterns:
    - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

4. オプションで、次のコマンドを入力して、**allowedSubjectPatterns** の識別名 (DN) を取得します。

```
$ openssl  x509 -in custom-cert.pem  -noout -subject
subject= /CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift
```

3.4. デフォルト INGRESS CONTROLLER の表示

Ingress Operator は OpenShift Dedicated 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 Dedicated インストールには、default という名前の **ingresscontroller** があります。これは、追加の Ingress Controller 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 Controller を表示します。

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

3.5. 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
```

3.6. INGRESS CONTROLLER ログの表示

Ingress Controller ログを表示できます。

手順

- Ingress Controller ログを表示します。

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c
<container_name>
```

3.7. INGRESS CONTROLLER ステータスの表示

特定の Ingress Controller のステータスを表示できます。

手順

- Ingress Controller のステータスを表示します。

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

3.8. カスタム INGRESS CONTROLLER の作成

クラスター管理者は、新規のカスタム Ingress Controller を作成できます。デフォルトの Ingress Controller は OpenShift Dedicated の更新時に変更される可能性があるため、カスタム Ingress コントローラーの作成は、クラスターの更新後も保持される設定を手動で維持する場合に役立ちます。

この例では、カスタム Ingress Controller の最小限の仕様を提供します。カスタム Ingress Controller をさらにカスタマイズするには、「Ingress Controller の設定」を参照してください。

前提条件

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

手順

1. カスタム **IngressController** オブジェクトを定義する YAML ファイルを作成します。

   custom-ingress-controller.yaml ファイルの例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <custom_name> 1
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: <custom-ingress-custom-certs> 2
  replicas: 1 3
  domain: <custom_domain> 4
```

1. **IngressController** オブジェクトのカスタム **名** を指定します。
2. カスタムワイルドカード証明書でシークレットの名前を指定します。

3 最小レプリカは ONE である必要があります

4 ドメイン名のドメインを指定します。IngressController オブジェクトで指定されるドメインと、証明書に使用されるドメインが一致する必要があります。たとえば、ドメイン値が "custom_domain.mycompany.com" の場合、証明書には SAN *.custom_domain.mycompany.com が必要です (***.** がドメインに追加されます)。

2. 以下のコマンドを実行してオブジェクトを作成します。

```
$ oc create -f custom-ingress-controller.yaml
```

3.9. INGRESS CONTROLLER の設定

3.9.1. 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 Controller がカスタム証明書を使用するように設定できます。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。ここで、証明書は信頼される認証局またはカスタム PKI で設定されたプライベートの信頼される認証局で署名されます。
- 証明書が以下の要件を満たしている必要があります。
  - 証明書が Ingress ドメインに対して有効化されている必要があります。
  - 証明書は拡張を使用して、**subjectAltName** 拡張を使用して、***.apps.ocp4.example.com** などのワイルドカードドメインを指定します。
- **IngressController** CR がなければなりません。デフォルトの CR を使用できます。

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

出力例

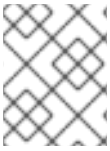
```
NAME      AGE
default   10m
```

注記

Intermediate 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 Intermediate 証明書をリスト表示します。

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 Controller はデプロイメントストラテジーを使用して再デプロイされます。

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

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

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

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

3. 更新が正常に行われていることを確認します。

```
$ echo Q |\
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null |\
  openssl x509 -noout -subject -issuer -enddate
```

ここでは、以下のようになります。

<domain>
クラスターのベースドメイン名を指定します。

出力例

```
subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
notAfter=May 10 08:32:45 2022 GM
```

ヒント

または、以下の YAML を適用してカスタムのデフォルト証明書を設定できます。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    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 Controller のデプロイメントを更新します。

3.9.2. カスタムデフォルト証明書の削除

管理者は、使用する Ingress Controller を設定したカスタム証明書を削除できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Ingress Controller のカスタムデフォルト証明書を設定している。

手順

- カスタム証明書を削除し、OpenShift Dedicated に付属の証明書を復元するには、次のコマンドを入力します。

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
  --type json -p $'- op: remove\n  path: /spec/defaultCertificate'
```

クラスターが新しい証明書設定を調整している間、遅延が発生する可能性があります。

検証

- 元のクラスター証明書が復元されたことを確認するには、次のコマンドを入力します。

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

ここでは、以下のようになります。

<domain>
クラスターのベースドメイン名を指定します。

出力例

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

3.9.3. Ingress Controller の自動スケーリング

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動的に対応するために自動でスケーリングできます。

以下の手順では、デフォルトの Ingress Controller をスケールアップする例を示します。

前提条件

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

- **cluster-admin** ロールを持つユーザーとして OpenShift Dedicated クラスターにアクセスできる。
- Custom Metrics Autoscaler Operator と関連する KEDA Controller がインストールされている。
  - この Operator は、Web コンソールで OperatorHub を使用してインストールできます。Operator をインストールしたら、**KedaController** のインスタンスを作成できます。

手順

1. 以下のコマンドを実行して、Thanos で認証するためのサービスアカウントを作成します。

```
$ oc create -n openshift-ingress-operator serviceaccount thanos && oc describe -n openshift-ingress-operator serviceaccount thanos
```

出力例

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-kfvf2
Mountable secrets:   thanos-dockercfg-kfvf2
Tokens:              <none>
Events:              <none>
```

2. 次のコマンドを使用して、サービスアカウントシークレットトークンを手動で作成します。

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: thanos-token
  namespace: openshift-ingress-operator
  annotations:
    kubernetes.io/service-account.name: thanos
type: kubernetes.io/service-account-token
EOF
```

3. サービスアカウントのトークンを使用して、**openshift-ingress-operator** namespace 内で **TriggerAuthentication** オブジェクトを定義します。
   a. **TriggerAuthentication** オブジェクトを作成し、**secret** 変数の値を **TOKEN** パラメーターに渡します。

```
$ oc apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-trigger-auth-prometheus
  namespace: openshift-ingress-operator
spec:
  secretTargetRef:
  - parameter: bearerToken
    name: thanos-token
```

```
      key: token
    - parameter: ca
      name: thanos-token
      key: ca.crt
EOF
```

4. Thanos からメトリクスを読み取るためのロールを作成して適用します。

   a. Pod およびノードからメトリクスを読み取る新規ロール **thanos-metrics-reader.yaml** を作成します。

      thanos-metrics-reader.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
  namespace: openshift-ingress-operator
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
```

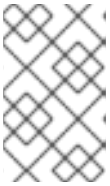
   b. 以下のコマンドを実行して新規ロールを適用します。

```
$ oc apply -f thanos-metrics-reader.yaml
```

5. 以下のコマンドを入力して、新しいロールをサービスアカウントに追加します。

```
$ oc adm policy -n openshift-ingress-operator add-role-to-user thanos-metrics-reader -z
thanos --role-namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-monitoring-view
-z thanos
```



注記

引数 **add-cluster-role-to-user** は、namespace 間のクエリーを使用する場合にのみ必要になります。以下の手順では、この引数を必要とする **kube-metrics** namespace からのクエリーを使用します。

6. デフォルトの Ingress Controller デプロイメントをターゲットにする新しい **ScaledObject** YAML ファイル **ingress-autoscaler.yaml** を作成します。

ScaledObject 定義の例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
  namespace: openshift-ingress-operator
spec:
  scaleTargetRef: 1
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 2
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 3
      namespace: openshift-ingress-operator 4
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' 5
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

1. 対象とするカスタムリソース。この場合、Ingress Controller。
2. オプション: レプリカの最大数。このフィールドを省略すると、デフォルトの最大値は 100 レプリカに設定されます。
3. **openshift-monitoring** namespace の Thanos サービスエンドポイント。
4. Ingress Operator namespace。
5. この式は、デプロイされたクラスターに存在するワーカーノードの数に対して評価されます。

重要

namespace 間クエリーを使用している場合は、**serverAddress** フィールドのポート 9092 ではなくポート 9091 をターゲットにする必要があります。また、このポートからメトリクスを読み取るには、昇格した権限が必要です。

7. 以下のコマンドを実行してカスタムリソース定義を適用します。

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

検証

- 以下のコマンドを実行して、デフォルトの Ingress Controller が **kube-state-metrics** クエリーによって返される値に一致するようにスケールアウトされていることを確認します。
  - **grep** コマンドを使用して、Ingress Controller の YAML ファイルでレプリカを検索します。

```
$ oc get -n openshift-ingress-operator ingresscontroller/default -o yaml | grep replicas:
```

出力例

```
  replicas: 3
```

  - **openshift-ingress** プロジェクトで Pod を取得します。

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

出力例

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
router-default-7b5df44ff-wwsth   2/2     Running   0          66s
```

3.9.4. Ingress Controller 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** リソースのスケーリングに使用されます。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。

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。

手順

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

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

出力例

```
2
```

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

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

出力例

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

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

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

出力例

```
3
```

ヒント

または、以下の YAML を適用して Ingress Controller を 3 つのレプリカにスケーリングすることもできます。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3 (1)
```

(1) 異なる数のレプリカが必要な場合は **replicas** 値を変更します。

3.9.5. Ingress アクセスロギングの設定

アクセスログを有効にするように Ingress Controller を設定できます。大量のトラフィックを受信しないクラスターがある場合、サイドカーにログインできます。クラスターのトラフィックが多い場合は、ロギングスタックの容量の超過を避けるために、または OpenShift Dedicated の外部のロギングインフラストラクチャーと統合するために、ログをカスタム syslog エンドポイントに転送できます。アクセスログの形式を指定することもできます。

コンテナーロギングは、既存の Syslog ロギングインフラストラクチャーがない場合や、Ingress Controller で問題を診断する際に短期間使用する場合に、低トラフィックのクラスターのアクセスログを有効にするのに役立ちます。

アクセスログが OpenShift Logging スタックの容量を超える可能性があるトラフィックの多いクラスターや、ロギングソリューションが既存の Syslog ロギングインフラストラクチャーと統合する必要のある環境では、syslog が必要です。Syslog のユースケースは重複する可能性があります。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

サイドカーへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。サイドカーコンテナーへのロギングを指定するには、**Container** **spec.logging.access.destination.type** を指定する必要があります。以下の例は、コンテナー **Container** の宛先に対してログ記録する Ingress Controller 定義です。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
```

- Ingress Controller をサイドカーに対してログを記録するように設定すると、Operator は Ingress Controller Pod 内に **logs** という名前のコンテナーを作成します。

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

出力例

```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080 0/0/1/0/1 200 142 - - --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

Syslog エンドポイントへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。Syslog エンドポイント宛先へのロギングを指定するには、**spec.logging.access.destination.type** に **Syslog** を指定する必要があります。宛先タイプが **Syslog** の場合、**spec.logging.access.destination.syslog.address** を使用して宛先エンドポイントも指定する必要があります。また、**spec.logging.access.destination.syslog.facility** を使用してファシリティーを指定できます。以下の例は、**Syslog** 宛先に対してログを記録する Ingress Controller の定義です。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```

注記

syslog 宛先ポートは UDP である必要があります。

syslog の宛先アドレスは IP アドレスにする必要があります。DNS ホスト名はサポートされていません。

特定のログ形式で Ingress アクセスロギングを設定します。

- **spec.logging.access.httpLogFormat** を指定して、ログ形式をカスタマイズできます。以下の例は、IP アドレスが 1.2.3.4 およびポート 10514 の **syslog** エンドポイントに対してログを記録する Ingress Controller の定義です。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
      httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

Ingress アクセスロギングを無効にします。

- Ingress アクセスロギングを無効にするには、**spec.logging** または **spec.logging.access** を空のままにします。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
```

```
spec:
  replicas: 2
  logging:
    access: null
```

サイドカーの使用時に Ingress Controller が HAProxy ログの長さを変更できるようにします。

- **spec.logging.access.destination.type: Syslog** を使用している場合は、**spec.logging.access.destination.syslog.maxLength** を使用します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514
```

- **spec.logging.access.destination.type: Container** を使用している場合は、**spec.logging.access.destination.container.maxLength** を使用します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
        container:
          maxLength: 8192
```

3.9.6. Ingress Controller スレッド数の設定

クラスター管理者は、スレッド数を設定して、クラスターが処理できる受信接続の量を増やすことができます。既存の Ingress Controller にパッチを適用して、スレッドの数を増やすことができます。

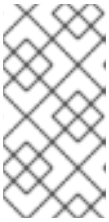
前提条件

- 以下では、Ingress Controller がすでに作成されていることを前提とします。

手順

- Ingress Controller を更新して、スレッド数を増やします。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"threadCount": 8}}}'
```

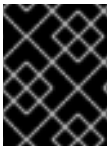


注記

大量のリソースを実行できるノードがある場合、**spec.nodePlacement.nodeSelector** を、意図されているノードの容量に一致するラベルで設定し、**spec.tuningOptions.threadCount** を随時高い値に設定します。

3.9.7. 内部ロードバランサーを使用するための Ingress Controller の設定

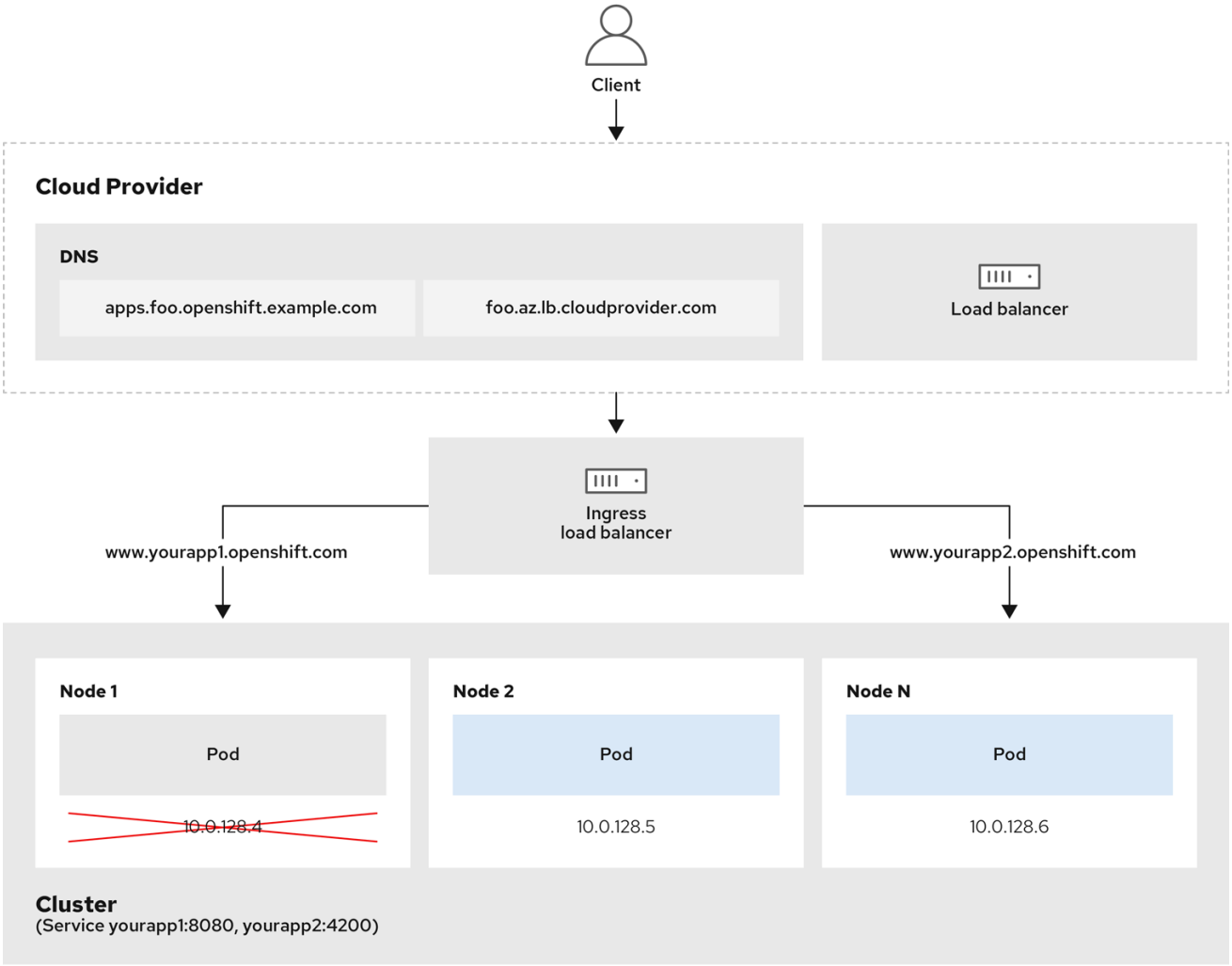
クラウドプラットフォームで Ingress Controller を作成する場合、Ingress Controller はデフォルトでパブリッククラウドロードバランサーによって公開されます。管理者は、内部クラウドロードバランサーを使用する Ingress Controller を作成できます。



重要

IngressController の **scope** を変更する場合は、カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更します。

図3.1 LoadBalancer の図



上の図は、OpenShift Dedicated の Ingress LoadBalancerService エンドポイント公開ストラテジーに関する次の概念を示しています。

- 負荷は、外部からクラウドプロバイダーのロードバランサーを使用するか、内部から OpenShift Ingress Controller Load Balancer を使用して、分散できます。
- ロードバランサーのシングル IP アドレスと、図にあるクラスターのように、8080 や 4200 といった馴染みのあるポートを使用することができます。
- 外部のロードバランサーからのトラフィックは、ダウンしたノードのインスタンスで記載されているように、Pod の方向に進められ、ロードバランサーが管理します。実装の詳細は、Kubernetes サービスドキュメント を参照してください。

前提条件

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

手順

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

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

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

2. 以下のコマンドを実行して、直前の手順で定義された Ingress Controller を作成します。

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

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

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

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

3.9.8. Ingress Controller のヘルスチェック間隔の設定

クラスター管理者は、ヘルスチェックの間隔を設定して、ルーターが連続する 2 回のヘルスチェックの間で待機する時間を定義できます。この値は、すべてのルートのデフォルトとしてグローバルに適用されます。デフォルト値は 5 秒です。

前提条件

- 以下では、Ingress Controller がすでに作成されていることを前提とします。

手順

- Ingress Controller を更新して、バックエンドヘルスチェックの間隔を変更します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```

注記

単一ルートの **healthCheckInterval** をオーバーライドするには、ルートアノテーション **router.openshift.io/haproxy.health.check.interval** を使用します

3.9.9. クラスターを内部に配置するためのデフォルト Ingress Controller の設定

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

重要

IngressController の **scope** を変更する場合は、カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更します。

前提条件

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

手順

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

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

3.9.10. ルートの受付ポリシーの設定

管理者およびアプリケーション開発者は、同じドメイン名を持つ複数の namespace でアプリケーションを実行できます。これは、複数のチームが同じホスト名で公開されるマイクロサービスを開発する組織を対象としています。

警告

複数の namespace での要求の許可は、namespace 間の信頼のあるクラスターに対してのみ有効にする必要があります。有効にしないと、悪意のあるユーザーがホスト名を乗っ取る可能性があります。このため、デフォルトの受付ポリシーは複数の namespace 間でのホスト名の要求を許可しません。

前提条件

- クラスター管理者の権限。

手順

- 以下のコマンドを使用して、**ingresscontroller** リソース変数の **.spec.routeAdmission** フィールドを編集します。

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

イメージコントローラー設定例

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

ヒント

または、以下の YAML を適用してルートの受付ポリシーを設定できます。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

3.9.11. ワイルドカードルートの使用

HAProxy Ingress Controller にはワイルドカードルートのサポートがあります。Ingress Operator は **wildcardPolicy** を使用して、Ingress Controller の **ROUTER_ALLOW_WILDCARD_ROUTES** 環境変数を設定します。

Ingress Controller のデフォルトの動作では、ワイルドカードポリシーの **None** (既存の **IngressController** リソースとの後方互換性がある) を持つルートを許可します。

手順

1. ワイルドカードポリシーを設定します。
   a. 以下のコマンドを使用して **IngressController** リソースを編集します。

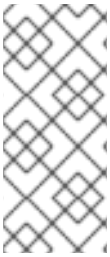
```
$ oc edit IngressController
```

   b. **spec** の下で、**wildcardPolicy** フィールドを **WildcardsDisallowed** または **WildcardsAllowed** に設定します。

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
```

3.9.12. HTTP ヘッダーの設定

OpenShift Dedicated は、HTTP ヘッダーを操作するさまざまな方法を提供します。ヘッダーを設定または削除する場合、Ingress Controller の特定のフィールドまたは個々のルートを使用して、リクエストヘッダーと応答ヘッダーを変更できます。ルートアノテーションを使用して特定のヘッダーを設定することもできます。ヘッダーを設定するさまざまな方法は、連携時に課題となる可能性があります。



注記

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

3.9.12.1. 優先順位

同じ HTTP ヘッダーを Ingress Controller とルートの両方で変更すると、HAProxy は、それがリクエストヘッダーであるか応答ヘッダーであるかに応じて、特定の方法でアクションの優先順位を付けます。

- HTTP 応答ヘッダーの場合、Ingress Controller で指定されたアクションは、ルートで指定されたアクションの後に実行されます。これは、Ingress Controller で指定されたアクションが優先されることを意味します。
- HTTP リクエストヘッダーの場合、ルートで指定されたアクションは、Ingress Controller で指定されたアクションの後に実行されます。これは、ルートで指定されたアクションが優先されることを意味します。

たとえば、クラスター管理者は、次の設定を使用して、Ingress Controller で X-Frame-Options 応答ヘッダーに値 **DENY** を設定します。

IngressController 仕様の例

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

ルート所有者は、クラスター管理者が Ingress Controller に設定したものと同じ応答ヘッダーを設定しますが、次の設定を使用して値 **SAMEORIGIN** を設定します。

Route 仕様の例

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
```

```
httpHeaders:
  actions:
    response:
    - name: X-Frame-Options
      action:
        type: Set
        set:
          value: SAMEORIGIN
```

IngressController 仕様と **Route** 仕様の両方で X-Frame-Options 応答ヘッダーが設定されている場合、特定のルートでフレームが許可されている場合でも、Ingress Controller のグローバルレベルでこのヘッダーに設定された値が優先されます。リクエストヘッダーの場合、**Route** 仕様の値が **IngressController** 仕様の値をオーバーライドします。

この優先順位付けは、**haproxy.config** ファイルで次のロジックが使用されるため発生します。このロジックでは、Ingress Controller がフロントエンドと見なされ、個々のルートがバックエンドと見なされます。フロントエンド設定に適用されるヘッダー値 **DENY** は、バックエンドで設定されている値 **SAMEORIGIN** で同じヘッダーをオーバーライドします。

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

さらに、Ingress Controller またはルートのいずれかで定義されたアクションは、ルートアノテーションを使用して設定された値をオーバーライドします。

3.9.12.2. 特殊なケースのヘッダー

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

表3.2 特殊な場合のヘッダー設定オプション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか

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

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

3.9.13. Ingress Controller での HTTP リクエストおよびレスポンスヘッダーの設定または削除

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

たとえば、相互 TLS を使用するようにクラスター上で実行されているアプリケーションを移行する場合があります。このような場合、お使いのアプリケーションで X-Forwarded-Client-Cert リクエストヘッダーをチェックする必要がありますが、OpenShift Dedicated のデフォルトの Ingress Controller は X-SSL-Client-Der リクエストヘッダーを提供します。

次の手順では、Ingress Controller を変更して X-Forwarded-Client-Cert リクエストヘッダーを設定し、X-SSL-Client-Der リクエストヘッダーを削除します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとして OpenShift Dedicated クラスターにアクセスできる。

手順

1. Ingress Controller リソースを編集します。

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. X-SSL-Client-Der HTTP リクエストヘッダーは X-Forwarded-Client-Cert HTTP リクエストヘッダーに置き換えます。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: 1
      request: 2
      - name: X-Forwarded-Client-Cert 3
        action:
          type: Set 4
          set:
            value: "%{+Q}[ssl_c_der,base64]" 5
      - name: X-SSL-Client-Der
        action:
          type: Delete
```

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

注記

HTTP 応答の動的ヘッダー値を設定する場合、サンプルフェッチャーとして **res.hdr** および **ssl_c_der** を使用できます。HTTP リクエストの動的ヘッダー値を設定する場合、許可されるサンプルフェッチャーは **req.hdr** および **ssl_c_der** です。リクエストとレスポンスの両方の動的値で、**lower** コンバーターと **Base64** コンバーターを使用できます。

3. 変更を適用するためにファイルを保存します。

3.9.14. X-Forwarded ヘッダーの使用

Forwarded および **X-Forwarded-For** を含む HTTP ヘッダーの処理方法に関するポリシーを指定するように HAProxy Ingress Controller を設定します。Ingress Operator は **HTTPHeaders** フィールドを使用して、Ingress Controller の **ROUTER_SET_FORWARDED_HEADERS** 環境変数を設定します。

手順

1. Ingress Controller 用に **HTTPHeaders** フィールドを設定します。

a. 以下のコマンドを使用して **IngressController** リソースを編集します。

```
$ oc edit IngressController
```

b. **spec** の下で、**HTTPHeaders** ポリシーフィールドを **Append**、**Replace**、**IfNone**、または **Never** に設定します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

使用例
クラスター管理者として、以下を実行できます。

- Ingress Controller に転送する前に、**X-Forwarded-For** ヘッダーを各リクエストに挿入する外部プロキシーを設定します。
  ヘッダーを変更せずに渡すように Ingress Controller を設定するには、**never** ポリシーを指定します。これにより、Ingress Controller はヘッダーを設定しなくなり、アプリケーションは外部プロキシーが提供するヘッダーのみを受信します。

- 外部プロキシーが外部クラスター要求を設定する **X-Forwarded-For** ヘッダーを変更せずに渡すように Ingress Controller を設定します。
  外部プロキシーを通過しない内部クラスター要求に **X-Forwarded-For** ヘッダーを設定するように Ingress Controller を設定するには、**if-none** ポリシーを指定します。外部プロキシー経由で HTTP 要求にヘッダーがすでに設定されている場合、Ingress Controller はこれを保持します。要求がプロキシーを通過していないためにヘッダーがない場合、Ingress Controller はヘッダーを追加します。

アプリケーション開発者として、以下を実行できます。

- **X-Forwarded-For** ヘッダーを挿入するアプリケーション固有の外部プロキシーを設定します。他の Route のポリシーに影響を与えずに、アプリケーションの Route 用にヘッダーを変更せずに渡すように Ingress Controller を設定するには、アプリケーションの Route にアノテーション **haproxy.router.openshift.io/set-forwarded-headers: if-none** または **haproxy.router.openshift.io/set-forwarded-headers: never** を追加します。

注記

Ingress Controller のグローバルに設定された値とは別に、**haproxy.router.openshift.io/set-forwarded-headers** アノテーションをルートごとに設定できます。

3.9.15. HTTP/2 Ingress 接続の有効化

HAProxy で透過的なエンドツーエンド HTTP/2 接続を有効にすることができます。これにより、アプリケーションの所有者は、単一接続、ヘッダー圧縮、バイナリーストリームなど、HTTP/2 プロトコル機能を使用できます。

個別の Ingress Controller またはクラスター全体について、HTTP/2 接続を有効にすることができます。

クライアントから HAProxy への接続について HTTP/2 の使用を有効にするために、ルートはカスタム証明書を指定する必要があります。デフォルトの証明書を使用するルートは HTTP/2 を使用することができません。この制限は、クライアントが同じ証明書を使用する複数の異なるルートに接続を再使用するなどの、接続の結合 (coalescing) の問題を回避するために必要です。

HAProxy からアプリケーション Pod への接続は、re-encrypt ルートのみに HTTP/2 を使用でき、edge-terminated ルートまたは非セキュアなルートには使用しません。この制限は、HAProxy が TLS 拡張である Application-Level Protocol Negotiation (ALPN) を使用してバックエンドで HTTP/2 の使用をネゴシエートするためにあります。そのため、エンドツーエンドの HTTP/2 はパススルーおよび re-encrypt 使用できますが、非セキュアなルートまたは edge-terminated ルートでは使用できません。

重要

パススルー以外のルートの場合、Ingress Controller はクライアントからの接続とは独立してアプリケーションへの接続をネゴシエートします。つまり、クライアントが Ingress Controller に接続して HTTP/1.1 をネゴシエートし、Ingress Controller は次にアプリケーションに接続して HTTP/2 をネゴシエートし、アプリケーションへの HTTP/2 接続を使用してクライアント HTTP/1.1 接続からの要求の転送を実行できます。Ingress Controller は WebSocket を HTTP/2 に転送できず、その HTTP/2 接続を WebSocket に対してアップグレードできないため、クライアントが後に HTTP/1.1 から WebSocket プロトコルに接続をアップグレードしようとすると問題が発生します。そのため、WebSocket 接続を受け入れることが意図されたアプリケーションがある場合、これは HTTP/2 プロトコルのネゴシエートを許可できないようにする必要があります。そうしないと、クライアントは WebSocket プロトコルへのアップグレードに失敗します。

手順

単一 Ingress Controller で HTTP/2 を有効にします。

- Ingress Controller で HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=true
```

<ingresscontroller_name> をアノテーションを付ける Ingress Controller の名前に置き換えます。

クラスター全体で HTTP/2 を有効にします。

- クラスター全体で HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-http2=true
```

ヒント

または、以下の YAML を適用してアノテーションを追加できます。

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "true"
```

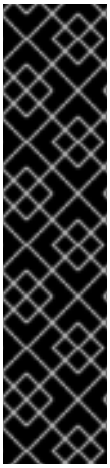
3.9.16. Ingress Controller の PROXY プロトコルの設定

クラスター管理者は、Ingress Controller が **HostNetwork**、**NodePortService**、または **Private** エンドポイント公開ストラテジータイプを使用する場合、PROXY プロトコル を設定できます。PROXY プロトコルにより、ロードバランサーは Ingress Controller が受信する接続の元のクライアントアドレスを保持することができます。元のクライアントアドレスは、HTTP ヘッダーのロギング、フィルタリング、および挿入を実行する場合に便利です。デフォルト設定では、Ingress Controller が受信する接続には、ロードバランサーに関連付けられるソースアドレスのみが含まれます。

警告

Keepalived Ingress 仮想 IP (VIP) を使用するクラウド以外のプラットフォーム上の installer-provisioned クラスターを備えたデフォルトの Ingress Controller は、PROXY プロトコルをサポートしていません。

PROXY プロトコルにより、ロードバランサーは Ingress Controller が受信する接続の元のクライアントアドレスを保持することができます。元のクライアントアドレスは、HTTP ヘッダーのロギング、フィルタリング、および挿入を実行する場合に便利です。デフォルト設定では、Ingress Controller が受信する接続には、ロードバランサーに関連付けられる送信元 IP アドレスのみが含まれます。



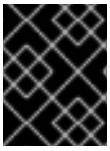
重要

パススルールート設定の場合、OpenShift Dedicated クラスター内のサーバーは元のクライアントソース IP アドレスを監視できません。元のクライアント送信元 IP アドレスを知る必要がある場合は、Ingress Controller の Ingress アクセスロギングを設定して、クライアント送信元 IP アドレスを表示できるようにします。

再暗号化およびエッジルートの場合、OpenShift Dedicated ルーターは **Forwarded** ヘッダーと **X-Forwarded-For** ヘッダーを設定し、アプリケーションワークロードがクライアントの送信元 IP アドレスをチェックできるようにします。

Ingress アクセスロギングの詳細は、「Ingress アクセスロギングの設定」を参照してください。

LoadBalancerService エンドポイント公開ストラテジータイプを使用する場合、Ingress Controller の PROXY プロトコルの設定はサポートされません。この制限があるのは、OpenShift Dedicated がクラウドプラットフォームで実行され、Ingress Controller がサービスロードバランサーを使用するように指定している場合に、Ingress Operator がロードバランサーサービスを設定し、ソースアドレスを保持するプラットフォーム要件に基づいて PROXY プロトコルを有効にするためです。



重要

OpenShift Dedicated と外部ロードバランサーの両方を、PROXY プロトコルまたは TCP のいずれかを使用するように設定する必要があります。

この機能は、クラウドデプロイメントではサポートされていません。この制限があるのは、OpenShift Dedicated がクラウドプラットフォームで実行され、Ingress Controller がサービスロードバランサーを使用するように指定している場合に、Ingress Operator がロードバランサーサービスを設定し、ソースアドレスを保持するプラットフォーム要件に基づいて PROXY プロトコルを有効にするためです。

重要

PROXY プロトコルまたは Transmission Control Protocol (TCP) を使用するには、OpenShift Dedicated と外部ロードバランサーの両方を設定する必要があります。

前提条件

- Ingress Controller を作成している。

手順

1. CLI で次のコマンドを入力して、Ingress Controller リソースを編集します。

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. PROXY 設定を設定します。

   - Ingress Controller が **HostNetwork** エンドポイント公開ストラテジータイプを使用する場合は、**spec.endpointPublishingStrategy.hostNetwork.protocol** サブフィールドを **PROXY** に設定します。

     PROXY への hostNetwork の設定例

```
# ...
  spec:
    endpointPublishingStrategy:
      hostNetwork:
        protocol: PROXY
      type: HostNetwork
# ...
```

   - Ingress Controller が **NodePortService** エンドポイント公開ストラテジータイプを使用する場合は、**spec.endpointPublishingStrategy.nodePort.protocol** サブフィールドを **PROXY** に設定します。

     PROXY へのサンプル nodePort 設定

```
# ...
  spec:
    endpointPublishingStrategy:
      nodePort:
        protocol: PROXY
      type: NodePortService
# ...
```

   - Ingress Controller が **Private** エンドポイント公開ストラテジータイプを使用する場合は、**spec.endpointPublishingStrategy.private.protocol** サブフィールドを **PROXY** に設定します。

     PROXY への private 設定のサンプル

```
# ...
  spec:
    endpointPublishingStrategy:
```

```
      private:
        protocol: PROXY
    type: Private
# ...
```

関連情報

- Ingress アクセスロギングの設定

3.9.17. appsDomain オプションを使用した代替クラスタードメインの指定

クラスター管理者は、**appsDomain** フィールドを設定して、ユーザーが作成したルートのデフォルトのクラスタードメインの代わりとなるものを指定できます。**appsDomain** フィールドは、**domain** フィールドで指定されているデフォルトの代わりに使用する OpenShift Dedicated のオプションのドメインです。代替ドメインを指定する場合、これは新規ルートのデフォルトホストを判別できるようにする目的でデフォルトのクラスタードメインを上書きします。

たとえば、所属企業の DNS ドメインを、クラスター上で実行されるアプリケーションのルートおよび ingress のデフォルトドメインとして使用できます。

前提条件

- OpenShift Dedicated クラスターをデプロイしている。
- **oc** コマンドラインインターフェイスがインストールされている。

手順

1. ユーザーが作成するルートに代替のデフォルトドメインを指定して **appsDomain** フィールドを設定します。
   a. Ingress **cluster** リソースを編集します。

      ```
      $ oc edit ingresses.config/cluster -o yaml
      ```

   b. YAML ファイルを編集します。

      test.example.com への appsDomain の設定例

      ```
      apiVersion: config.openshift.io/v1
      kind: Ingress
      metadata:
        name: cluster
      spec:
        domain: apps.example.com
        appsDomain: <test.example.com>
      ```
      (1) (2)

      (1) デフォルトドメインを指定します。インストール後にデフォルトドメインを変更することはできません。

      (2) オプション: アプリケーションルートに使用する OpenShift Dedicated インフラストラクチャーのドメイン。デフォルトの接頭辞である **apps** の代わりに、**test** のような別の接頭辞を使用できます。

2. ルートを公開し、ルートドメインの変更を確認して、既存のルートに、**appsDomain** フィールドで指定したドメイン名が含まれていることを確認します。

注記

ルートを公開する前に **openshift-apiserver** がローリング更新を終了するのを待機します。

a. ルートを公開します。

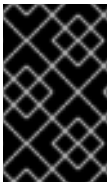
```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

出力例

```
$ oc get routes
NAME              HOST/PORT                                   PATH   SERVICES         PORT
TERMINATION   WILDCARD
hello-openshift   hello_openshift-<my_project>.test.example.com
hello-openshift   8080-tcp                None
```

3.9.18. HTTP ヘッダーケースの変換

HAProxy では、デフォルトで HTTP ヘッダー名を小文字化します。たとえば、**Host: xyz.com** は **host: xyz.com** に変更されます。レガシーアプリケーションが HTTP ヘッダー名の大文字を認識する場合、Ingress Controller の **spec.httpHeaders.headerNameCaseAdjustments** API フィールドを、修正されるまでレガシーアプリケーションに対応するソリューションに使用します。



重要

OpenShift Dedicated には HAProxy 2.8 が含まれています。このバージョンの Web ベースのロードバランサーに更新する場合は、必ずクラスターの設定ファイルに **spec.httpHeaders.headerNameCaseAdjustments** セクションを追加してください。

クラスター管理者は、**oc patch** コマンドを入力するか、Ingress Controller YAML ファイルの **HeaderNameCaseAdjustments** フィールドを設定して HTTP ヘッダーのケースを変換できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

- **oc patch** コマンドを使用して、HTTP ヘッダーを大文字にします。

  a. 次のコマンドを実行して、HTTP ヘッダーを **host** から **Host** に変更します。

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}}'
```

b. アノテーションをアプリケーションに適用できるように、**Route** リソースの YAML ファイルを作成します。

my-application という名前のルートの例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: <application_name>
  namespace: <application_name>
# ...
```

(1) Ingress コントローラーが指定どおりに **host** リクエストヘッダーを調整できるように、**haproxy.router.openshift.io/h1-adjust-case** を設定します。

- Ingress Controller YAML 設定ファイルで **HeaderNameCaseAdjustments** フィールドを設定して調整を指定します。

a. 次の Ingress Controller YAML ファイルの例では、適切にアノテーションが付けられたルートへの HTTP/1 リクエストの **host** ヘッダーを **Host** に調整します。

Ingress Controller YAML のサンプル

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
    - Host
```

b. 次のルートの例では、**haproxy.router.openshift.io/h1-adjust-case** アノテーションを使用して HTTP レスポンスヘッダー名の大文字と小文字の調整を有効にします。

ルート YAML のサンプル

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

(1) **haproxy.router.openshift.io/h1-adjust-case** を true に設定します。

3.9.19. ルーター圧縮の使用

特定の MIME タイプに対してルーター圧縮をグローバルに指定するように HAProxy Ingress Controller を設定します。**mimeTypes**変数を使用して、圧縮が適用される MIME タイプの形式を定義できます。タイプは、アプリケーション、イメージ、メッセージ、マルチパート、テキスト、ビデオ、または "X-" で始まるカスタムタイプです。MIME タイプとサブタイプの完全な表記を確認するには、RFC1341を参照してください。

注記

圧縮用に割り当てられたメモリーは、最大接続数に影響を与える可能性があります。さらに、大きなバッファーを圧縮すると、正規表現による負荷が多い場合や正規表現のリストが長い場合など、レイテンシーが発生する可能性があります。

すべての MIME タイプが圧縮から利点を得るわけではありませんが、HAProxy は、指示された場合でもリソースを使用して圧縮を試みます。一般に、html、css、js などのテキスト形式は圧縮から利点を得ますが、イメージ、音声、ビデオなどのすでに圧縮済みの形式は、圧縮に時間とリソースが費やされるわりに利点はほぼありません。

手順

1. Ingress Controller の**httpCompression**フィールドを設定します。
   a. 以下のコマンドを使用して **IngressController** リソースを編集します。

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```

   b. **spec**で、**httpCompression** ポリシーフィールドを**mimeTypes** に設定し、圧縮を適用する必要がある MIME タイプのリストを指定します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
    mimeTypes:
    - "text/html"
    - "text/css; charset=utf-8"
    - "application/json"
     ...
```

3.9.20. ルーターメトリクスの公開

デフォルトで、HAProxy ルーターメトリクスをデフォルトの stats ポート (1936) に Prometheus 形式で公開できます。Prometheus などの外部メトリクス収集および集約システムは、HAProxy ルーターメメトリクスにアクセスできます。HAProxy ルーターメトリクスは、HTML およびコンマ区切り値 (CSV) 形式でブラウザーに表示できます。

前提条件

- ファイアウォールを、デフォルトの stats ポート (1936) にアクセスするように設定している。

手順

手順

1. 次のコマンドを実行して、ルーター Pod 名を取得します。

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

出力例

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-76bfffb66c-46qwp   1/1     Running   0          11h
```

2. ルーター Pod が **/var/lib/haproxy/conf/metrics-auth/statsUsername** および **/var/lib/haproxy/conf/metrics-auth/statsPassword** ファイルに保存しているルーターのユーザー名およびパスワードを取得します。

   a. 次のコマンドを実行して、ユーザー名を取得します。

```
$ oc rsh <router_pod_name> cat metrics-auth/statsUsername
```

   b. 次のコマンドを実行して、パスワードを取得します。

```
$ oc rsh <router_pod_name> cat metrics-auth/statsPassword
```

3. 次のコマンドを実行して、ルーター IP およびメトリクス証明書を取得します。

```
$ oc describe pod <router_pod>
```

4. つぎのコマンドを実行して、Prometheus 形式で未加工の統計情報を取得します。

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

5. 次のコマンドを実行して、安全にメトリクスにアクセスします。

```
$ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
```

6. 次のコマンドを実行して、デフォルトの stats ポート (1936) にアクセスします。

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

例3.1 出力例

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
```

```
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service="
"} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""}
0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. ブラウザーで以下の URL を入力して、stats ウィンドウを起動します。

```
http://<user>:<password>@<router_IP>:<stats_port>
```

8. オプション: ブラウザーに次の URL を入力して、CSV 形式で統計情報を取得します。

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

3.9.21. HAProxy エラーコードの応答ページのカスタマイズ

クラスター管理者は、503、404、またはその両方のエラーページにカスタムのエラーコード応答ページを指定できます。HAProxy ルーターは、アプリケーション Pod が実行していない場合や、要求された URL が存在しない場合に 404 エラーページを提供する 503 エラーページを提供します。たとえば、503 エラーコードの応答ページをカスタマイズする場合は、アプリケーション Pod が実行していないときにページが提供されます。また、デフォルトの 404 エラーコード HTTP 応答ページは、誤ったルートまたは存在しないルートについて HAProxy ルーターによって提供されます。

カスタムエラーコードの応答ページは config map に指定し、Ingress Controller にパッチを適用されます。config map キーには、**error-page-503.http** と **error-page-404.http** の 2 つの利用可能なファイル名があります。

カスタムの HTTP エラーコードの応答ページは、HAProxy HTTP エラーページ設定のガイドライン に従う必要があります。以下は、デフォルトの OpenShift Dedicated HAProxy ルーターの http 503 エラーコード応答ページ の例です。デフォルトのコンテンツを、独自のカスタムページを作成するためのテンプレートとして使用できます。

デフォルトで、HAProxy ルーターは、アプリケーションが実行していない場合や、ルートが正しくないまたは存在しない場合に 503 エラーページのみを提供します。このデフォルトの動作は、OpenShift Dedicated 4.8 以前の動作と同じです。HTTP エラーコード応答をカスタマイズするための config map が提供されておらず、カスタム HTTP エラーコード応答ページを使用している場合、ルーターはデフォルトの 404 または 503 エラーコード応答ページを提供します。

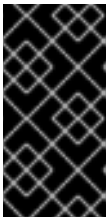
注記

OpenShift Dedicated のデフォルトの 503 エラーコードページをカスタマイズのテンプレートとして使用する場合、ファイル内のヘッダーで CRLF 改行コードを使用できるエディターが必要になります。

手順

1. **openshift-config** に **my-custom-error-code-pages** という名前の config map を作成します。

```
$ oc -n openshift-config create configmap my-custom-error-code-pages \
--from-file=error-page-503.http \
--from-file=error-page-404.http
```



重要

カスタムエラーコードの応答ページに適した形式を指定しない場合は、ルーター Pod が停止します。この停止を解決するには、config map を削除するか、修正し、影響を受けるルーター Pod を削除して、正しい情報で再作成できるようにします。

2. Ingress Controller にパッチを適用し、名前を指定して **my-custom-error-code-pages** config map を参照します。

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":
{"httpErrorCodePages":{"name":"my-custom-error-code-pages"}}}' --type=merge
```

Ingress Operator は、**openshift-config** namespace から **openshift-ingress** namespace に **my-custom-error-code-pages** config map をコピーします。Operator は、**openshift-ingress** namespace のパターン **<your_ingresscontroller_name>-errorpages** に従って config map に名前を付けます。

3. コピーを表示します。

```
$ oc get cm default-errorpages -n openshift-ingress
```

出力例

```
NAME                 DATA   AGE
default-errorpages     2    25s  1
```

1 **default** の Ingress Controller カスタムリソース (CR) にパッチが適用されているため、config map 名の例は **default-errorpages** です。

4. カスタムエラー応答ページを含む config map がルーターボリュームにマウントされることを確認します。config map キーは、カスタム HTTP エラーコード応答を持つファイル名です。

- 503 カスタム HTTP カスタムエラーコード応答の場合:

```
$ oc -n openshift-ingress rsh <router_pod> cat
/var/lib/haproxy/conf/error_code_pages/error-page-503.http
```

- 404 カスタム HTTP カスタムエラーコード応答の場合:

```
$ oc -n openshift-ingress rsh <router_pod> cat
/var/lib/haproxy/conf/error_code_pages/error-page-404.http
```

検証

カスタムエラーコード HTTP 応答を確認します。

1. テストプロジェクトおよびアプリケーションを作成します。

```
$ oc new-project test-ingress
```

```
$ oc new-app django-psql-example
```

2. 503 カスタム http エラーコード応答の場合:
   a. アプリケーションのすべての Pod を停止します。
   b. 以下の curl コマンドを実行するか、ブラウザーでルートのホスト名にアクセスします。

```
$ curl -vk <route_hostname>
```

3. 404 カスタム http エラーコード応答の場合:
   a. 存在しないルートまたは正しくないルートにアクセスします。
   b. 以下の curl コマンドを実行するか、ブラウザーでルートのホスト名にアクセスします。

```
$ curl -vk <route_hostname>
```

4. **errorfile** 属性が **haproxy.config** ファイルで適切にあるかどうかを確認します。

```
$ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep errorfile
```

3.9.22. Ingress Controller の最大接続数の設定

クラスター管理者は、OpenShift ルーターデプロイメントの同時接続の最大数を設定できます。既存の Ingress Controller にパッチを適用して、接続の最大数を増やすことができます。

前提条件

- 以下では、Ingress Controller が作成済みであることを前提とします。

手順

- Ingress Controller を更新して、HAProxy の最大接続数を変更します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"maxConnections": 7500}}}'
```

警告

spec.tuningOptions.maxConnections の値を現在のオペレーティングシステムの制限よりも大きく設定すると、HAProxy プロセスは開始しません。このパラメーターの詳細は、「Ingress Controller 設定パラメーター」セクションの表を参照してください。

3.10. OPENSHIFT DEDICATED INGRESS OPERATOR の設定

以下の表は、Ingress Operator のコンポーネントおよび、Red Hat Site Reliability Engineers (SRE) が OpenShift Dedicated クラスターでこのコンポーネントを管理するかどうかについて詳しく説明します。

表3.3 Ingress Operator の責務チャート

Ingress コンポーネント	管理	デフォルト設定？
Ingress コントローラーのスケーリング	SRE	はい
Ingress Operator のスレッド数	SRE	はい
Ingress コントローラーのアクセスロギング	SRE	はい
Ingress Controller のシャード化	SRE	はい
Ingress コントローラールートの受付ポリシー	SRE	はい
Ingress コントローラーのワイルドカードルート	SRE	はい
Ingress コントローラー X-Forwarded ヘッダー	SRE	はい
Ingress コントローラールートの圧縮	SRE	はい

第4章 OPENSHIFT DEDICATED クラスターのネットワーク検証

OpenShift Dedicated クラスターを既存の Virtual Private Cloud (VPC) にデプロイするとき、またはクラスターに新しいサブネットを持つ追加のマシンプールを作成するときに、ネットワーク検証チェックが自動的に実行します。このチェックによりネットワーク設定が検証され、エラーが強調表示されるため、デプロイメント前に設定の問題を解決できます。

ネットワーク検証チェックを手動で実行して、既存のクラスターの設定を検証することもできます。

4.1. OPENSHIFT DEDICATED クラスターのネットワーク検証について

OpenShift Dedicated クラスターを既存の Virtual Private Cloud (VPC) にデプロイするか、クラスターに新しいサブネットを持つ追加のマシンプールを作成すると、ネットワーク検証が自動的に実行します。これは、デプロイメント前に設定の問題を特定して解決するのに役立ちます。

Red Hat OpenShift Cluster Manager を使用してクラスターをインストールする準備をする場合は、**Virtual Private Cloud (VPC) subnet settings**ページのサブネット ID フィールドにサブネットを入力した後に自動チェックが実行します。

新しいサブネットを持つマシンプールをクラスターに追加すると、マシンプールがプロビジョニングされる前に、自動ネットワーク検証によってサブネットがチェックされ、ネットワーク接続が利用可能であることが確認されます。

自動ネットワーク検証が完了すると、レコードがサービスログに送信されます。レコードには、ネットワーク設定エラーを含む検証チェックの結果が示されます。特定された問題をデプロイメント前に解決すると、デプロイメントが成功する可能性が高くなります。

既存のクラスターに対してネットワーク検証を手動で実行することもできます。これにより、設定を変更した後にクラスターのネットワーク設定を検証できます。ネットワーク検証チェックを手動で実行する手順は、**ネットワーク検証を手動で実行する** を参照してください。

4.2. ネットワーク検証チェックの範囲

ネットワーク検証には、次の各要件のチェックが含まれます。

- 親の Virtual Private Cloud (VPC) がある。
- 指定されたすべてのサブネットは VPC に属している。
- VPC で **enableDnsSupport** が有効になっている。
- VPC で **enableDnsHostnames** が有効になっている。
- Egress は、AWS ファイアウォールの前提条件 セクションで指定されている必要なドメインとポートの組み合わせで利用できます。

4.3. 自動ネットワーク検証のバイパス

既知のネットワーク設定の問題がある OpenShift Dedicated クラスターを既存の Virtual Private Cloud (VPC) にデプロイする場合は、自動ネットワーク検証を回避できます。

クラスターの作成時にネットワーク検証を回避すると、クラスターのサポートステータスは制限されます。インストール後、問題を解決してからネットワーク検証を手動で実行できます。制限付きサポートステータスは、検証が成功すると削除されます。

Red Hat OpenShift Cluster Manager を使用して既存の VPC にクラスターをインストールする時に、**Virtual Private Cloud (VPC) subnet settings**ページで **Bypass network verification** を選択することで自動検証を回避できます。

4.4. ネットワーク検証を手動で実行する

Red Hat OpenShift Cluster Manager を使用して、既存の OpenShift Dedicated クラスターのネットワーク検証チェックを手動で実行できます。

前提条件

- 既存の OpenShift Dedicated クラスターがある。
- クラスター所有者になっているか、クラスター編集者のロールがある。

手順

1. OpenShift Cluster Manager に移動し、クラスターを選択します。
2. **Actions** ドロップダウンメニューから **Verify networking** を選択します。

第5章 クラスター全体のプロキシーの設定

既存の Virtual Private Cloud (VPC) を使用している場合は、OpenShift Dedicated クラスターのインストール中またはクラスターのインストール後に、クラスター全体のプロキシーを設定できます。プロキシーを有効にすると、コアクラスターコンポーネントはインターネットへの直接アクセスを拒否されますが、プロキシーはユーザーのワークロードには影響しません。

注記

クラウドプロバイダー API への呼び出しを含め、クラスターシステムの egress トラフィックのみがプロキシーされます。

プロキシーは、Customer Cloud Subscription (CCS) モデルを使用する OpenShift Dedicated クラスターに対してのみ有効にできます。

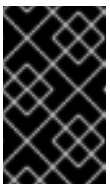
クラスター全体のプロキシーを使用する場合は、責任をもってクラスターへのプロキシーの可用性を確保してください。プロキシーが利用できなくなると、クラスターの正常性とサポート性に影響を与える可能性があります。

5.1. クラスター全体のプロキシーを設定するための前提条件

クラスター全体のプロキシーを設定するには、次の要件を満たす必要があります。これらの要件は、インストール中またはインストール後にプロキシーを設定する場合に有効です。

一般要件

- クラスターの所有者である。
- アカウントには十分な権限がある。
- クラスターに既存の Virtual Private Cloud (VPC) がある。
- クラスターに Customer Cloud Subscription (CCS) モデルを使用している。
- プロキシーは、クラスターの VPC および VPC のプライベートサブネットにアクセスできる。またクラスターの VPC および VPC のプライベートサブネットからもアクセスできる。
- 次のエンドポイントが VPC エンドポイントに追加されている。
  - **ec2.<aws_region>.amazonaws.com**
  - **elasticloadbalancing.<aws_region>.amazonaws.com**
  - **s3.<aws_region>.amazonaws.com**
    これらのエンドポイントは、ノードから AWS EC2 API への要求を完了するために必要です。プロキシーはノードレベルではなくコンテナーレベルで機能するため、これらの要求を AWS プライベートネットワークを使用して AWS EC2 API にルーティングする必要があります。プロキシーサーバーの許可リストに EC2 API のパブリック IP アドレスを追加するだけでは不十分です。



重要

クラスター全体のプロキシーを使用する場合は、**s3.<aws_region>.amazonaws.com** エンドポイントを **Gateway** のタイプとして設定する必要があります。

ネットワーク要件

- プロキシーが出力トラフィックを再登録する場合は、ドメインとポートの組み合わせに対する除外を作成する必要があります。次の表は、これらの例外のガイダンスを示しています。
  - プロキシーは、以下の OpenShift URL の再暗号化を除外する必要があります。

住所	プロトコル/ポート	機能
observatorium-mst.api.openshift.com	https/443	必須。Managed OpenShift 固有の Telemetry に使用されます。
sso.redhat.com	https/443	https://cloud.redhat.com/openshift サイトでは、sso.redhat.com からの認証を使用してプルシークレットをダウンロードし、Red Hat SaaS ソリューションを使用してサブスクリプション、クラスターインベントリー、チャージバックレポートなどのモニタリングを行います。

  - プロキシーでは、次のサイトリライアビリティーエンジニアリング (SRE) および管理 URL の再暗号化を除外する必要があります。

住所	プロトコル/ポート	機能
***.osdsecuritylogs.splunkcloud.com** OR **inputs1.osdsecuritylogs.splunkcloud.cominputs2.osdsecuritylogs.splunkcloud.cominputs4.osdsecuritylogs.splunkcloud.cominputs5.osdsecuritylogs.splunkcloud.cominputs6.osdsecuritylogs.splunkcloud.cominputs7.osdsecuritylogs.splunkcloud.cominputs8.osdsecuritylogs.splunkcloud.cominputs9.osdsecuritylogs.splunkcloud.cominputs10.osdsecuritylogs.splunkcloud.cominputs11.osdsecuritylogs.splunkcloud.cominputs12.osdsecuritylogs.splunkcloud.cominputs13.osdsecuritylogs.splunkcloud.cominputs14.osdsecuritylogs.splunkcloud.cominputs15.osdsecuritylogs.splunkcloud.com**	tcp/9997	ログベースのアラートについて Red Hat SRE が使用するロギング転送エンドポイントとして splunk-forwarder-operator によって使用されます。
http-inputs-osdsecuritylogs.splunkcloud.com	https/443	ログベースのアラートについて Red Hat SRE が使用するロギング転送エンドポイントとして splunk-forwarder-operator によって使用されます。

関連情報

- Customer Cloud Subscription (CCS) モデルを使用する OpenShift Dedicated クラスターのインストールの前提条件については、AWS での Customer Cloud Subscription または GCP での Customer Cloud Subscription を参照してください。

5.2. 追加の信頼バンドルに対する責任

追加の信頼バンドルを指定する場合は、以下の要件を満たす必要があります。

- 追加の信頼バンドルの内容が有効であることを確認する
- 追加の信頼バンドルに含まれる中間証明書を含む証明書の有効期限が切れていないことを確認する
- 追加の信頼バンドルに含まれる証明書の有効期限を追跡して必要な更新を実行する
- 更新された追加の信頼バンドルを使用してクラスター設定を更新する

5.3. インストール中にプロキシーを設定する

OpenShift Dedicated with Customer Cloud Subscription (CCS) クラスターを既存の Virtual Private Cloud (VPC) にインストールするときに、HTTP または HTTPS プロキシーを設定できます。Red Hat OpenShift Cluster Manager を使用して、インストール中にプロキシーを設定できます。

5.4. OPENSHIFT CLUSTER MANAGER を使用したインストール時のプロキシーの設定

OpenShift Dedicated クラスターを既存の Virtual Private Cloud (VPC) にインストールする場合、Red Hat OpenShift Cluster Manager を使用して、インストール中にクラスター全体の HTTP または HTTPS プロキシーを有効にすることができます。プロキシーは、Customer Cloud Subscription (CCS) モデルを使用するクラスターに対してのみ有効にできます。

インストールの前に、クラスターがインストールされている VPC からプロキシーにアクセスできることを確認する必要があります。プロキシーは VPC のプライベートサブネットからもアクセスできる必要があります。

OpenShift Cluster Manager を使用してインストール中にクラスター全体のプロキシーを設定する詳細な手順については、**CCS を使用した AWS でのクラスターの作成** または **CCS を使用した GCP でのクラスターの作成** を参照してください。

関連情報

- CCS を使用した AWS でのクラスターの作成
- CCS を使用した GCP でのクラスターの作成

5.5. インストール後のプロキシーの設定

OpenShift Dedicated with Customer Cloud Subscription (CCS) クラスターを既存の Virtual Private Cloud (VPC) にインストールした後、HTTP または HTTPS プロキシーを設定できます。Red Hat OpenShift Cluster Manager を使用して、インストール後にプロキシーを設定できます。

5.6. OPENSHIFT CLUSTER MANAGER を使用したインストール後のプロキシーの設定

Red Hat OpenShift Cluster Manager を使用して、Virtual Private Cloud (VPC) の既存の OpenShift Dedicated クラスターにクラスター全体のプロキシー設定を追加できます。プロキシーは、Customer Cloud Subscription (CCS) モデルを使用するクラスターに対してのみ有効にできます。

OpenShift Cluster Manager を使用して、既存のクラスター全体のプロキシー設定を更新することもできます。たとえば、プロキシーのネットワークアドレスを更新するか、プロキシーの認証局のいずれかが期限切れになる場合は追加の信頼バンドルを置き換える必要がある場合があります。

重要

クラスターはプロキシー設定をコントロールプレーンおよびコンピュートノードに適用します。設定の適用時に、各クラスターノードは一時的にスケジュール不可能な状態になり、そのワークロードがドレイン (解放) されます。プロセスの一環として各ノードが再起動されます。

前提条件

- Customer Cloud Subscription (CCS) モデルを使用する OpenShift Dedicated クラスターがある。
- クラスターが VPC にデプロイされている。

手順

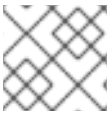
1. OpenShift Cluster Manager に移動し、クラスターを選択します。
2. **Networking** ページの **Virtual Private Cloud (VPC)**セクションで、**Edit cluster-wide proxy** をクリックします。
3. **Edit cluster-wide proxy** ページで、プロキシー設定の詳細を指定します。
   a. 次のフィールドの少なくとも 1 つに値を入力します。
      - 有効な **HTTP proxy URL** を指定します。
      - 有効な **HTTPS proxy URL** を指定します。
      - **Additional trust bundle** フィールドに、PEM でエンコードされた X.509 証明書バンドルを指定します。既存の信頼バンドルファイルを置き換える場合は、**Replace file** を選択してフィールドを表示します。このバンドルはクラスターノードの信頼済み証明書ストアに追加されます。TLS 検査プロキシーを使用する場合は、プロキシーのアイデンティティー証明書が Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルからの認証局によって署名されない限り、追加の信頼バンドルファイルが必要です。この要件は、プロキシーが透過的であるか、**http-proxy** および **https-proxy** 引数を使用して明示的な設定を必要とするかに関係なく適用されます。
   b. **Confirm** をクリックします。

検証

- **Networking** ページの **Virtual Private Cloud (VPC)**セクションで、クラスターのプロキシー設定が想定どおりであることを確認します。

第6章 CIDR 範囲の定義

次の CIDR 範囲には、重複しない範囲を指定する必要があります。



注記

クラスターの作成後にマシンの CIDR 範囲を変更することはできません。

サブネット CIDR 範囲を指定するときは、サブネット CIDR 範囲が定義済みの Machine CIDR 内にあることを確認してください。クラスターがホストされているプラットフォームに応じて、サブネット CIDR 範囲ですべての対象ワークロードに十分な IP アドレスが許可されていることを確認する必要があります。

重要

OpenShift Dedicated 4.14 以降のバージョンのデフォルトネットワークプロバイダーである OVN-Kubernetes は、内部的に IP アドレス範囲 (**100.64.0.0/16**、**169.254.169.0/29**、**100.88.0.0/16**、**fd98::/64**、**fd69::/125**、および **fd97::/64**) を使用します。クラスターで OVN-Kubernetes を使用する場合は、クラスターまたはインフラストラクチャー内の他の CIDR 定義にこれらの IP アドレス範囲を含めないでください。

OpenShift Dedicated 4. **17 以降のバージョンの場合、クラスターは IPv4 に 169.254.0.0** /17 を使用し、IPv6 用に **fd69::/112** をデフォルトのマスカレードサブネットとして使用します。ユーザーはこれらの範囲も回避する必要があります。アップグレードされたクラスターの場合は、デフォルトのマスカレードサブネットに変更がありません。

6.1. MACHINE CIDR

マシンの Classless Inter-Domain Routing (CIDR) フィールドでは、マシンまたはクラスターノードの IP アドレス範囲を指定する必要があります。この範囲には、仮想プライベートクラウド (VPC) サブネットのすべての CIDR アドレス範囲が含まれている必要があります。サブネットは連続している必要があります。単一のアベイラビリティーゾーンデプロイメントでは、サブネット接頭辞 **/25** を使用した 128 アドレスの最小 IP アドレス範囲がサポートされます。サブネット接頭辞 **/24** を使用する最小アドレス範囲 256 アドレスの範囲は、複数のアベイラビリティーゾーンを使用するデプロイメントでサポートされます。

デフォルトは **10.0.0.0/16** です。この範囲は、接続されているネットワークと競合しないようにする必要があります。

6.2. SERVICE CIDR

Service CIDR フィールドで、サービスの IP アドレス範囲を指定する必要があります。必須ではありませんが、クラスター間でアドレスブロックを同じにすることが推奨されます。これにより、IP アドレスの競合が発生することはありません。範囲は、ワークロードに対応するのに十分な大きさである必要があります。アドレスブロックは、クラスター内からアクセスする外部サービスと重複してはいけません。デフォルトは **172.30.0.0/16** です。

6.3. POD CIDR

Pod CIDR フィールドで、Pod の IP アドレス範囲を指定する必要があります。

必須ではありませんが、クラスター間でアドレスブロックを同じにすることが推奨されます。これによ

り、IP アドレスの競合が発生することはありません。範囲は、ワークロードに対応するのに十分な大きさである必要があります。アドレスブロックは、クラスター内からアクセスする外部サービスと重複してはいけません。デフォルトは **10.128.0.0/14** です。

6.4. ホスト接頭辞

Host Prefix フィールドで、個々のマシンにスケジュールされた Pod に割り当てられたサブネット接頭辞の長さを指定する必要があります。ホスト接頭辞は、各マシンの Pod IP アドレスプールを決定します。

例えば、ホスト接頭辞を **/23** に設定した場合、各マシンには Pod CIDR アドレス範囲から **/23** のサブネットが割り当てられます。デフォルトは **/23** で、クラスターノード数は 512、ノードあたりの Pod 数は 512 となっていますが、いずれも弊社がサポートする最大値を超えています。

第7章 ネットワークセキュリティー

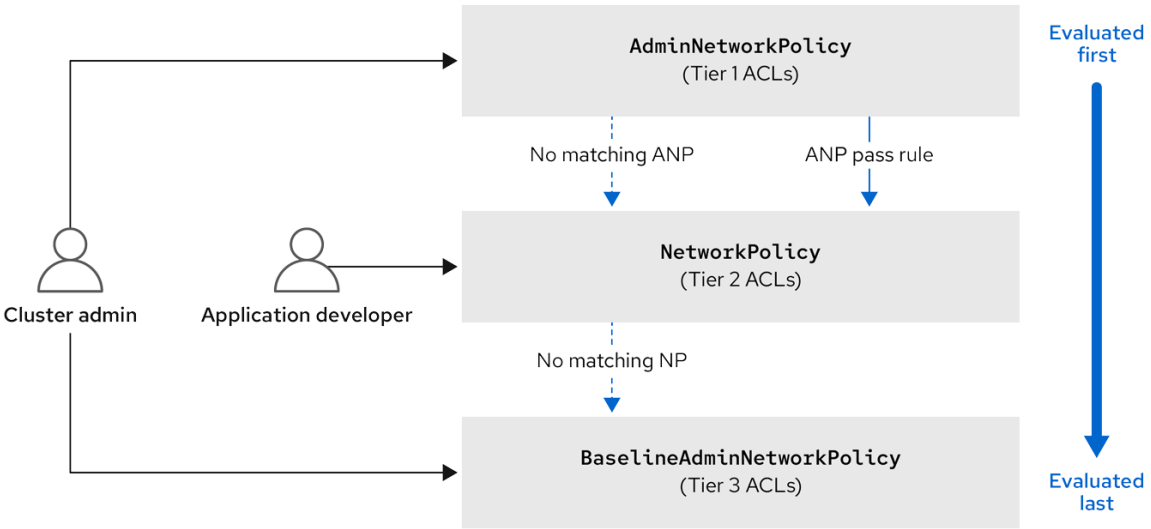
7.1. ネットワークポリシー API について

Kubernetes は、ネットワークセキュリティーの強化に使用できる 2 つの機能を提供します。機能の 1 つは、ユーザーによるネットワークポリシーの適用を可能にする **NetworkPolicy** API です。これは主にアプリケーション開発者と namespace テナント向けの機能で、namespace スコープのポリシーを作成して namespace を保護することを目的としています。

2 番目の機能は **AdminNetworkPolicy** で、**AdminNetworkPolicy** (ANP) API と **BaselineAdminNetworkPolicy** (BANP) API の 2 つの API で構成されています。ANP と BANP は、クラスターおよびネットワーク管理者向けの機能で、クラスタースコープのポリシーを作成してクラスター全体を保護することを目的としています。クラスター管理者は、ANP を使用すると、**NetworkPolicy** オブジェクトよりも優先されるオーバーライド不可能なポリシーを適用できます。BANP を使用すると、**NetworkPolicy** オブジェクトを使用して必要に応じてユーザーがオーバーライドできるオプションのクラスタースコープのネットワークポリシールールをセットアップおよび適用できます。ANP、BANP、およびネットワークポリシーを一緒に使用すると、管理者がクラスターのセキュリティー保護に使用できる完全なマルチテナント分離を実現できます。

OpenShift Dedicated の OVN-Kubernetes CNI は、アクセス制御リスト (ACL) の階層を使用してこれらのネットワークポリシーを実装し、それらを評価して適用します。ACL は、階層 1 から階層 3 まで降順で評価されます。

階層 1 では **AdminNetworkPolicy** (ANP) オブジェクトを評価します。階層 2 では **NetworkPolicy** オブジェクトを評価します。階層 3 では **BaselineAdminNetworkPolicy** (BANP) オブジェクトを評価します。



最初に ANP が評価されます。一致が ANP **allow** または **deny** ルールである場合、クラスター内の既存の **NetworkPolicy** および **BaselineAdminNetworkPolicy** (BANP) オブジェクトは評価からスキップされます。一致が ANP の **pass** ルールの場合、評価が ACL 階層 1 から階層 2 に進み、そこで **NetworkPolicy** ポリシーが評価されます。トラフィックに一致する **NetworkPolicy** がない場合、評価は Tier 2 ACL から Tier 3 ACL に移動し、そこで BANP が評価されます。

7.1.1. AdminNetworkPolicy と NetworkPolicy カスタムリソースの主な違い

次の表は、クラススコープの **AdminNetworkPolicy** API と namespace スコープの **NetworkPolicy** API の主な違いを説明しています。

ポリシーの要素	AdminNetworkPolicy	NetworkPolicy
対象ユーザー	クラスター管理者または同等の権限	namespace の所有者
スコープ	クラスター	Namespace を使用
トラフィックのドロップ	明示的な **Deny** アクションをルールとして設定することでサポートされます。	ポリシー作成時に暗黙的に **Deny** 分離することでサポートされます。
トラフィックの委譲	**Pass** アクションをルールとして設定することでサポートされます。	該当なし
トラフィックの許可	明示的に **Allow** アクションをルールとして設定することでサポートされます。	すべてのルールに対するデフォルトのアクションは allow です。
ポリシー内のルールの優先順位	ANP 内で表示される順序によって異なります。ルールの位置が高いほど、優先順位が高くなります。	ルールは追加できます。
ポリシーの優先順位	ANP 間では、**priority** フィールドによって評価の順序が設定されます。優先順位の数字が低いほど、ポリシーの優先順位が高くなります。	ポリシー間にポリシー順序はありません。
機能の優先順位	最初に Tier 1 ACL を介して評価され、最後に BANP が Tier 3 ACL を介して評価されます。	ANP の後、BANP の前に適用され、ACL の Tier 2 で評価されます。
Pod 選択の一致	namespace 間で異なるルールを適用できます。	1 つの namespace 内の Pod に異なるルールを適用できます。
クラスターの Egress トラフィック	**nodes** と **networks** ピアを介してサポートされます。	受け入れられた CIDR 構文とともに **ipBlock** フィールドを使用してサポートされます。
クラスター Ingress トラフィック	サポート対象外	サポート対象外
完全修飾ドメイン名 (FQDN) ピアサポート	サポート対象外	サポート対象外

ポリシーの要素	AdminNetworkPolicy	NetworkPolicy
namespace セレクター	**namespaces.matchLabels** フィールドを使用した namespace の高度な選択をサポートします	**namespaceSelector** フィールドを使用したラベルベースでの namespace の選択をサポートします

7.2. ネットワークポリシー

7.2.1. ネットワークポリシーについて

開発者は、クラスター内の Pod へのトラフィックを制限するネットワークポリシーを定義できます。

7.2.1.1. ネットワークポリシーについて

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の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、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

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。

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

警告

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

ネットワークポリシーは、ローカルホストまたは常駐ノードからのトラフィックをブロックすることはできません。

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

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

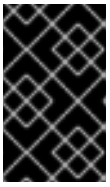
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
```

```
spec:
  podSelector: {}
  ingress: []
```

- 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 Controller からの接続のみを許可するには、次の **NetworkPolicy** オブジェクトを追加します。

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

- プロジェクト内の Pod からの接続のみを受け入れます。



重要

同じ namespace 内の **hostNetwork** Pod からの Ingress 接続を許可するには、**allow-from-hostnetwork** ポリシーと **allow-same-namespace** ポリシーを一緒に適用する必要があります。

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。

7.2.1.1.1. allow-from-router ネットワークポリシーの使用

次の **NetworkPolicy** を使用して、ルーターの設定に関係なく外部トラフィックを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" (1)
```

```
  podSelector: {}
  policyTypes:
  - Ingress
```

1 **policy-group.network.openshift.io/ingress:""** ラベルは OVN-Kubernetes をサポートします。

7.2.1.1.2. allow-from-hostnetwork ネットワークポリシーの使用

次の **allow-from-hostnetwork NetworkPolicy** オブジェクトを追加して、ホストネットワーク Pod からのトラフィックを転送します。

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

7.2.1.2. OpenShift SDN を使用したネットワークポリシー最適化

ネットワークポリシーを使用して、namespace 内でラベルで相互に区別される Pod を分離します。

NetworkPolicy オブジェクトを単一 namespace 内の多数の個別 Pod に適用することは効率的ではありません。Pod ラベルは IP レベルには存在しないため、ネットワークポリシーは、**podSelector** で選択されるすべての Pod 間のすべてのリンクに関する別個の Open vSwitch (OVS) フロールールを生成します。

たとえば、仕様の **podSelector** および **NetworkPolicy** オブジェクト内の Ingress **podSelector** のそれぞれが 200 Pod に一致する場合、40,000 (200*200) OVS フロールールが生成されます。これにより、ノードの速度が低下する可能性があります。

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

- namespace を使用して分離する必要のある Pod のグループを組み込み、OVS フロールールの数を減らします。
  namespace 全体を選択する **NetworkPolicy** オブジェクトは、**namespaceSelector** または空の **podSelector** を使用して、namespace の VXLAN 仮想ネットワーク ID (VNID) に一致する単一の OVS フロールールのみを生成します。

- 分離する必要のない Pod は元の namespace に維持し、分離する必要のある Pod は 1 つ以上の異なる namespace に移します。

- 追加のターゲット設定された namespace 間のネットワークポリシーを作成し、分離された Pod から許可する必要のある特定のトラフィックを可能にします。

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

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

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

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

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

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

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

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

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
spec:
  podSelector:
    matchLabels:
```

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

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

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

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

7.2.1.4. 次のステップ

- ネットワークポリシーの作成

7.2.2. ネットワークポリシーの作成

admin ロールを持つユーザーは、namespace のネットワークポリシーを作成できます。

7.2.2.1. サンプル NetworkPolicy オブジェクト

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

■

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

1 NetworkPolicy オブジェクトの名前。

2 ポリシーが適用される Pod を説明するセレクター。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入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namaspace にある Pod を照合して検索します。

4 トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

7.2.2.2. CLI を使用したネットワークポリシーの作成

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

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. ポリシールールを作成します。

a. **<policy_name>.yaml** ファイルを作成します。

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーファイル名を指定します。

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

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

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

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

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

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

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

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

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

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

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

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

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーファイル名を指定します。

<namespace>

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

出力例

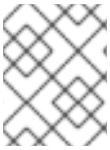
```
networkpolicy.networking.k8s.io/deny-by-default created
```

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接作成できます。

7.2.2.3. デフォルトの全拒否ネットワークポリシーの作成

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



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

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

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

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

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

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

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

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

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

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

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

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。

- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

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

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

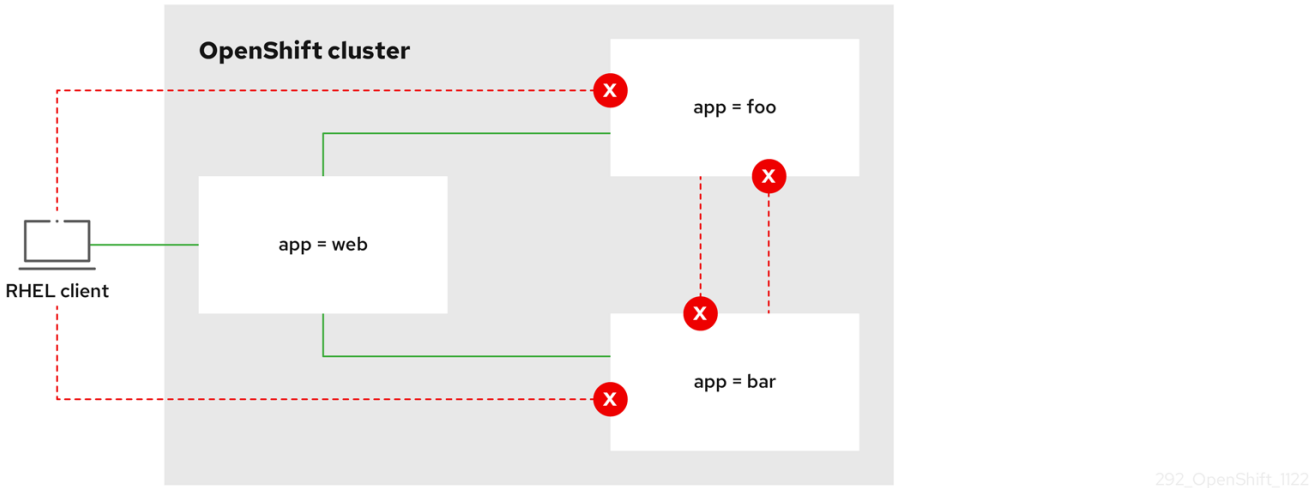
2. 次のコマンドを入力して、ポリシーを適用します。

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

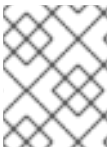
出力例

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

このポリシーは、次の図に示すように、外部トラフィックを含むすべてのリソースからのトラフィックを許可します。



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



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

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

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

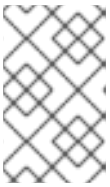
手順

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

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

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

(2) すべての namespace のすべての Pod を選択します。



注記

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

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

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

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

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

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

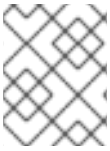
3. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

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

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

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

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



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

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

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

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

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

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

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

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

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

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

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

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

```
$ oc create namespace prod
```

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

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

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

```
$ oc create namespace dev
```

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

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

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

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

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

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

予想される出力

```
wget: download timed out
```

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

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

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

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

予想される出力

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

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

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

7.2.2.7. OpenShift Cluster Manager を使用したネットワークポリシーの作成

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

前提条件

- OpenShift Cluster Manager にログインしている。
- OpenShift Dedicated クラスターを作成している。
- クラスターにアイデンティティープロバイダーを設定している。
- 設定したアイデンティティープロバイダーにユーザーアカウントを追加している。
- OpenShift Dedicated クラスター内にプロジェクトを作成しました。

手順

1. OpenShift Cluster Manager で、アクセスするクラスターをクリックします。
2. **コンソールを開く** をクリックして、OpenShift Web コンソールに移動します。
3. アイデンティティープロバイダーをクリックし、クラスターにログインするためのクレデンシャルを指定します。

4. 管理者の観点から、**Networking** の下の **NetworkPolicies** をクリックします。

5. **NetworkPolicy の作成** をクリックします。

6. **ポリシー名** フィールドにポリシーの名前を入力します。

7. オプション: このポリシーが 1 つ以上の特定の Pod にのみ適用される場合は、特定の Pod のラベルとセレクターを指定できます。特定の Pod を選択しない場合、このポリシーはクラスター上のすべての Pod に適用されます。

8. オプション: **Deny all ingress traffic** または **Deny all egress traffic** チェックボックスを使用して、すべてのイングレストラフィックとエグレストラフィックをブロックできます。

9. イングレスルールとエグレスルールの任意の組み合わせを追加して、承認するポート、名前空間、または IP ブロックを指定することもできます。

10. Ingress ルールをポリシーに追加します。

    a. **Add ingress rule** を選択して新規ルールを設定します。このアクションにより、受信トラフィックを制限する方法を指定できる **Add allowed source** ドロップダウンメニューを含む新しい **Ingress ルール** 行が作成されます。ドロップダウンメニューでは、Ingress トラフィックを制限する 3 つのオプションを利用できます。

       - **Allow pods from the same namespace**では、空間内の Pod へのトラフィックが制限されます。namespace に Pod を指定できますが、このオプションは空のままにすると namespace の Pod からのすべてのトラフィックを許可します。

       - **Allow pods from inside the cluster**では、ポリシーと同じクラスター内の Pod へのトラフィックが制限されます。インバウンドトラフィックを許可する名前空間と Pod を指定できます。このオプションを空白のままにすると、このクラスター内のすべての名前空間と Pod からのインバウンドトラフィックが許可されます。

       - **IP ブロックによるピアの許可** は、指定された Classless Inter-Domain Routing (CIDR) IP ブロックからのトラフィックを制限します。例外オプションを使用して、特定の IP をブロックできます。CIDR フィールドを空白のままにすると、すべての外部ソースからのすべてのインバウンドトラフィックが許可されます。

    b. すべての受信トラフィックをポートに制限できます。ポートを追加しない場合、トラフィックはすべてのポートにアクセスできます。

11. ネットワークポリシーにエグレスルールを追加します。

    a. **Add egress rule** 選択して、新しいルールを設定します。このアクションにより、送信トラフィックを制限する方法を指定できる **Add allowed destination**"* する * ドロップダウンメニューを含む新しい **Egress rule** 行が作成されます。ドロップダウンメニューには、下りトラフィックを制限する 3 つのオプションがあります。

       - **Allow pods from the same namespace**では、同じ namespace 内の Pod へのトラフィックが制限されます。namespace に Pod を指定できますが、このオプションは空のままにすると namespace の Pod からのすべてのトラフィックを許可します。

       - **Allow pods from inside the cluster**では、ポリシーと同じクラスター内の Pod へのトラフィックが制限されます。アウトバウンドトラフィックを許可する namespace および Pod を指定できます。このオプションを空白のままにすると、このクラスター内のすべての名前空間と Pod からのアウトバウンドトラフィックが許可されます。

       - **Allow peers by IP block** すると、指定された CIDR IP ブロックからのトラフィックが制限されます。例外オプションを使用して、特定の IP をブロックできます。CIDR

フィールドを空白のままにすると、すべての外部ソースからのすべてのアウトバウンドが許可されます。

b. すべてのアウトバウンドトラフィックをポートに制限できます。ポートを追加しない場合、トラフィックはすべてのポートにアクセスできます。

7.2.3. ネットワークポリシーの表示

admin ロールを持つユーザーは、namespace のネットワークポリシーを表示できます。

7.2.3.1. サンプル NetworkPolicy オブジェクト

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

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

1 NetworkPolicy オブジェクトの名前。

2 ポリシーが適用される Pod を説明するセレクター。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入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namaspace にある Pod を照合して検索します。

4 トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

7.2.3.2. CLI を使用したネットワークポリシーの表示

namespace のネットワークポリシーを検査できます。

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意のネットワークポリシーを表示できます。

前提条件

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

- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

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

```
$ oc get networkpolicy
```

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

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

ここでは、以下のようになります。

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

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

以下に例を示します。

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

oc describe コマンドの出力

```
Name:         allow-same-namespace
Namespace:    ns1
Created on:   2021-05-24 22:28:56 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: <none>
  Not affecting egress traffic
  Policy Types: Ingress
```

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接表示できます。

7.2.3.3. OpenShift Cluster Manager を使用したネットワークポリシーの表示

Red Hat OpenShift Cluster Manager でネットワークポリシーの設定の詳細を表示できます。

前提条件

- OpenShift Cluster Manager にログインしている。
- OpenShift Dedicated クラスターを作成している。
- クラスターにアイデンティティープロバイダーを設定している。
- 設定したアイデンティティープロバイダーにユーザーアカウントを追加している。
- ネットワークポリシーを作成しました。

手順

1. OpenShift Cluster Manager Web コンソールの **Administrator** パースペクティブから、**Networking** の下にある **NetworkPolicies** をクリックします。
2. 表示するネットワークポリシーを選択します。
3. **ネットワークポリシー** の詳細ページで、関連付けられたすべての Ingress および egress ルールを表示できます。
4. ネットワークポリシーの詳細で **YAML** を選択して、ポリシー設定を YAML 形式で表示します。

注記

これらのポリシーの詳細のみを表示できます。これらのポリシーは編集できません。

7.2.4. ネットワークポリシーの削除

admin ロールを持つユーザーは、namespace からネットワークポリシーを削除できます。

7.2.4.1. CLI を使用したネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意のネットワークポリシーを削除できます。

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

- ネットワークポリシーが存在する namespace で作業している。

手順

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

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

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

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

出力例

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

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールの **Actions** メニューのポリシーから、クラスターの任意の namespace でネットワークポリシーを直接削除できます。

7.2.4.2. OpenShift Cluster Manager を使用したネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。

前提条件

- OpenShift Cluster Manager にログインしている。
- OpenShift Dedicated クラスターを作成している。
- クラスターにアイデンティティープロバイダーを設定している。
- 設定したアイデンティティープロバイダーにユーザーアカウントを追加している。

手順

1. OpenShift Cluster Manager Web コンソールの **Administrator** パースペクティブから、**Networking** の下にある **NetworkPolicies** をクリックします。
2. ネットワークポリシーを削除するには、次のいずれかの方法を使用します。
   - **ネットワークポリシー** テーブルからポリシーを削除します。
     a. **ネットワークポリシー** テーブルから、削除するネットワークポリシーの行にあるスタックメニューを選択し、**NetworkPolicy の削除** をクリックします。

- 個々のネットワークポリシーの詳細から **アクション** ドロップダウンメニューを使用してポリシーを削除します。

  a. ネットワークポリシーの **アクション** ドロップダウンメニューをクリックします。

  b. メニューから **Delete NetworkPolicy** を選択します。

7.2.5. ネットワークポリシーを使用したマルチテナント分離の設定

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

注記

このセクションで説明するようにネットワークポリシーを設定すると、{product-name} の以前のバージョンの OpenShift SDN のマルチテナントモードと同様のネットワーク分離が実現します。

7.2.5.1. ネットワークポリシーを使用したマルチテナント分離の設定

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

前提条件

- クラスターが、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインまたは OpenShift SDN ネットワークプラグインなど) を使用している。このモードは OpenShift SDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. 以下の **NetworkPolicy** オブジェクトを作成します。

   a. **allow-from-openshift-ingress** という名前のポリシー:

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

注記

policy-group.network.openshift.io/ingress: ""は、OpenShift SDN の推奨の namespace セレクターラベルです。**network.openshift.io/policy-group: ingress** namespace セレクターラベルを使用できますが、これはレガシーラベルです。

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

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

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

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

d. **allow-from-kube-apiserver-operator** という名前のポリシー:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
```

```
  policyTypes:
  - Ingress
EOF
```

詳細は、新規の New **kube-apiserver-operator** webhook controller validating health of webhook を参照してください。

2. オプション: 以下のコマンドを実行し、ネットワークポリシー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 describe networkpolicy
```

出力例

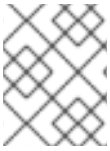
```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

第8章 OVN-KUBERNETES ネットワークプラグイン

8.1. OVN-KUBERNETES ネットワークプラグインについて

OpenShift Dedicated クラスターは、Pod およびサービスネットワークに仮想化ネットワークを使用します。

Red Hat OpenShift Networking の一部である OVN-Kubernetes ネットワークプラグインは、OpenShift Dedicated のデフォルトのネットワークプロバイダーです。OVN-Kubernetes は Open Virtual Network (OVN) をベースとしており、オーバーレイベースのネットワーク実装を提供します。OVN-Kubernetes プラグインを使用するクラスターは、各ノードで Open vSwitch (OVS) も実行します。OVN は、宣言ネットワーク設定を実装するように各ノードで OVS を設定します。



注記

OVN-Kubernetes は、OpenShift Dedicated およびシングルノード OpenShift デプロイメントのデフォルトのネットワークソリューションです。

OVS プロジェクトから生まれた OVN-Kubernetes は、オープンフロールールなど、同じコンストラクトの多くを使用して、パケットがネットワークを通過する方法を決定します。詳細は、Open Virtual Network の Web サイト を参照してください。

OVN-Kubernetes は、仮想ネットワーク設定を **OpenFlow** ルールに変換する OVS 用の一連のデーモンです。**OpenFlow** は、ネットワークスイッチおよびルーターと通信するためのプロトコルであり、ネットワークデバイス上のネットワークトラフィックのフローをリモートで制御する手段を提供し、ネットワーク管理者がネットワークトラフィックのフローを設定、管理、および監視できるようにします。

OVN-Kubernetes は、**OpenFlow** では利用できない高度な機能をさらに提供します。OVN は、分散仮想ルーター、分散論理スイッチ、アクセス制御、DHCP および DNS をサポートします。OVN は、オープンフローと同等のロジックフロー内に分散仮想ルーターを実装します。たとえば、ネットワーク上に DHCP リクエストを送信する Pod がある場合、Pod はそのブロードキャストを送信して DHCP アドレスを探します。また、そのパケットに一致するロジックフロールールが存在し、応答としてゲートウェイ、DNS サーバー、IP アドレスなどを提供します。

OVN-Kubernetes は、各ノードでデーモンを実行します。すべてのノードで実行されるデータベースおよび OVN コントローラー用のデーモンセットがあります。OVN コントローラーは、ネットワークプロバイダーの機能 (Egress IP、ファイアウォール、ルーター、ハイブリッドネットワーク、IPSEC 暗号化、IPv6、ネットワークポリシー、ネットワークポリシーログ、ハードウェアオフロード、およびマルチキャスト) をサポートするために、ノード上で Open vSwitch デーモンをプログラムします。

8.1.1. OVN-Kubernetes の目的

OVN-Kubernetes ネットワークプラグインは、Open Virtual Network (OVN) を使用してネットワークトラフィックフローを管理する、オープンソースのフル機能の Kubernetes CNI プラグインです。OVN はコミュニティーで開発され、ベンダーに依存しないネットワーク仮想化ソリューションです。OVN-Kubernetes ネットワークプラグインは次のテクノロジーを使用します。

- ネットワークトラフィックフローを管理するための OVN。
- Ingress ルールおよび Egress ルールを含む Kubernetes ネットワークポリシーのサポートとログ。
- ノード間にオーバーレイネットワークを作成するための、Virtual Extensible LAN (VXLAN) ではなく、Generic Network Virtualization Encapsulation (Geneve) プロトコル。

OVN-Kubernetes ネットワークプラグインは、次の機能をサポートしています。

- Linux と Microsoft Windows の両方のワークロードを実行できるハイブリッドクラスター。この環境は **ハイブリッドネットワーキング** と呼ばれます。
- ホストの中央処理装置 (CPU) から互換性のあるネットワークカードおよびデータ処理装置 (DPU) へのネットワークデータ処理のオフロード。これは **ハードウェアオフロード** と呼ばれます。
- ベアメタル、VMware vSphere、IBM Power®、IBM Z®、および RHOSP プラットフォーム上の IPv4 プライマリーデュアルスタックネットワーク。
- ベアメタルプラットフォーム上の IPv6 シングルスタックネットワーキング。
- ベアメタル、VMware vSphere、または RHOSP プラットフォーム上で実行しているクラスター用の IPv6 プライマリーデュアルスタックネットワーク。
- Egress ファイアウォールデバイスと Egress IP アドレス。
- リダイレクトモードで動作する Egress ルーターデバイス。
- クラスター内通信の IPsec 暗号化。

8.1.2. OVN-Kubernetes IPv6 とデュアルスタックの制限

OVN-Kubernetes ネットワークプラグインには、次の制限があります。

- デュアルスタックネットワークに設定されたクラスターでは、IPv4 と IPv6 の両方のトラフィックがデフォルトゲートウェイとして同じネットワークインターフェイスを使用する必要があります。この要件が満たされない場合には、**ovnkube-node** デーモンセットのホストにある Pod は、**CrashLoopBackOff** 状態になります。**oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml** のようなコマンドで Pod を表示すると、以下の出力のように、**status** フィールドにデフォルトゲートウェイに関する複数のメッセージが表示されます。

  ```
  I1006 16:09:50.985852   60651 helper_linux.go:73] Found default gateway interface br-ex 192.168.127.1
  I1006 16:09:50.985923   60651 helper_linux.go:73] Found default gateway interface ens4 fe80::5054:ff:febe:bcd4
  F1006 16:09:50.985939   60651 ovnkube.go:130] multiple gateway interfaces detected: br-ex ens4
  ```

  唯一の解決策は、両方の IP ファミリーがデフォルトゲートウェイに同じネットワークインターフェイスを使用するように、ホストネットワークを再設定することです。

- デュアルスタックネットワーク用に設定されたクラスターの場合、IPv4 と IPv6 の両方のルーティングテーブルにデフォルトゲートウェイが含まれている必要があります。この要件が満たされない場合には、**ovnkube-node** デーモンセットのホストにある Pod は、**CrashLoopBackOff** 状態になります。**oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml** のようなコマンドで Pod を表示すると、以下の出力のように、**status** フィールドにデフォルトゲートウェイに関する複数のメッセージが表示されます。

  ```
  I0512 19:07:17.589083  108432 helper_linux.go:74] Found default gateway interface br-ex 192.168.123.1
  F0512 19:07:17.589141  108432 ovnkube.go:133] failed to get default gateway interface
  ```

唯一の解決策として、両方の IP ファミリーにデフォルトゲートウェイが含まれるようにホストネットワークを再設定できます。

8.1.3. セッションアフィニティー

セッションアフィニティーは、Kubernetes **Service** オブジェクトに適用される機能です。<service_VIP>:<Port> に接続するたびに、トラフィックが常に同じバックエンドに負荷分散されるようにする場合は、**セッションアフィニティー** を使用できます。クライアントの IP アドレスに基づいてセッションアフィニティーを設定する方法など、詳細は、セッションアフィニティー を参照してください。

セッションアフィニティーのスティッキネスタイムアウト
OpenShift Dedicated の OVN-Kubernetes ネットワークプラグインは、最後のパケットに基づいてクライアントからのセッションのスティッキネスタイムアウトを計算します。たとえば、**curl** コマンドを 10 回実行すると、スティッキーセッションタイマーは最初のパケットではなく 10 番目のパケットから開始します。その結果、クライアントが継続的にサービスに接続している場合でも、セッションがタイムアウトすることはありません。タイムアウトは、**timeoutSeconds** パラメーターで設定された時間、サービスがパケットを受信しなかった場合に開始されます。

第9章 ルートの作成

9.1. ルート設定

9.1.1. HTTP ベースのルートの作成

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

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 管理者としてログインしている。
- あるポートを公開する Web アプリケーションと、そのポートでトラフィックをリッスンする TCP エンドポイントがあります。

手順

1. 次のコマンドを実行して、**hello-openshift** というプロジェクトを作成します。

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

2. 以下のコマンドを実行してプロジェクトに Pod を作成します。

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

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

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

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

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

検証

- 作成した **route** リソースを確認するには、次のコマンドを実行します。

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

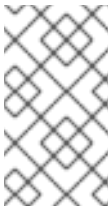
1 この例では、ルートの名前は **hello-openshift** です。

上記で作成したセキュアでないルートの YAML 定義

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

1 **<Ingress_Domain>** はデフォルトの Ingress ドメイン名です。**ingresses.config/cluster** オブジェクトはインストール中に作成され、変更できません。別のドメインを指定する場合は、**appsDomain** オプションを使用して別のクラスタードメインを指定できます。

2 **targetPort** は、このルートが指すサービスによって選択される Pod のターゲットポートです。



注記

デフォルトの Ingress ドメインを表示するには、以下のコマンドを実行します。

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

9.1.2. 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 Controller 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
```

9.1.3. HTTP Strict Transport Security

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

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

クラスター管理者は、以下を実行するために HSTS を設定できます。

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

重要

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

9.1.3.1. ルートごとの HTTP Strict Transport Security の有効化

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

前提条件

- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

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

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

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

注記

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

アノテーションで設定されたルートの例

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

1. 必須。**max-age** は、HSTS ポリシーが有効な期間 (秒単位) を測定します。**0** に設定すると、これはポリシーを無効にします。
2. オプション: **includeSubDomains** は、クライアントに対し、ホストのすべてのサブドメインにホストと同じ HSTS ポリシーを持つ必要があることを指示します。
3. オプション: **max-age** が 0 より大きい場合、**preload** を **haproxy.router.openshift.io/hsts_header** に追加し、外部サービスがこのサイトをそれぞれの HSTS プリロードリストに含めることができます。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザーはこれらのリストを使用し、サイトと対話する前でも HTTPS 経由で通信できるサイトを判別できます。**preload** を設定していない場合、ブラウザーはヘッダーを取得するために、HTTPS を介してサイトと少なくとも 1 回対話している必要があります。

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

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

前提条件

- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

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

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

ヒント

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

ルートごとに HSTS を無効にする例

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

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

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

検証

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

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

出力例

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

9.1.4. Cookie の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 Controller に対し、セッションを処理しているエンドポイントを示し、クライアント要求が Cookie を使用して同じ Pod にルーティングされるようにします。

注記

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

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

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

手順

1. 指定される cookie 名でルートにアノテーションを付けます。

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

ここでは、以下のようになります。

<route_name>
Pod の名前を指定します。

<cookie_name>
cookie の名前を指定します。

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

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

2. 変数でルートのホスト名を取得します。

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

ここでは、以下のようになります。

<route_name>
Pod の名前を指定します。

3. cookie を保存してからルートにアクセスします。

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

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

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

9.1.5. パスベースのルート

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

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

表9.1 ルートの可用性

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

パスが 1 つでセキュリティー保護されていないルート

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

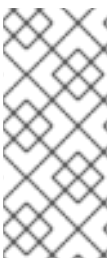
1 パスは、パスベースのルートに唯一追加される属性です。

注記

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

9.1.6. HTTP ヘッダーの設定

OpenShift Dedicated は、HTTP ヘッダーを操作するさまざまな方法を提供します。ヘッダーを設定または削除する場合、Ingress Controller の特定のフィールドまたは個々のルートを使用して、リクエストヘッダーと応答ヘッダーを変更できます。ルートアノテーションを使用して特定のヘッダーを設定することもできます。ヘッダーを設定するさまざまな方法は、連携時に課題となる可能性があります。



注記

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

9.1.6.1. 優先順位

同じ HTTP ヘッダーを Ingress Controller とルートの両方で変更すると、HAProxy は、それがリクエストヘッダーであるか応答ヘッダーであるかに応じて、特定の方法でアクションの優先順位を付けます。

- HTTP 応答ヘッダーの場合、Ingress Controller で指定されたアクションは、ルートで指定されたアクションの後に実行されます。これは、Ingress Controller で指定されたアクションが優先されることを意味します。
- HTTP リクエストヘッダーの場合、ルートで指定されたアクションは、Ingress Controller で指定されたアクションの後に実行されます。これは、ルートで指定されたアクションが優先されることを意味します。

たとえば、クラスター管理者は、次の設定を使用して、Ingress Controller で X-Frame-Options 応答ヘッダーに値 **DENY** を設定します。

IngressController 仕様の例

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

ルート所有者は、クラスター管理者が Ingress Controller に設定したものと同じ応答ヘッダーを設定しますが、次の設定を使用して値 **SAMEORIGIN** を設定します。

Route 仕様の例

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

IngressController 仕様と **Route** 仕様の両方で X-Frame-Options 応答ヘッダーが設定されている場合、特定のルートでフレームが許可されている場合でも、Ingress Controller のグローバルレベルでこのヘッダーに設定された値が優先されます。リクエストヘッダーの場合、**Route** 仕様の値が **IngressController** 仕様の値をオーバーライドします。

この優先順位付けは、**haproxy.config** ファイルで次のロジックが使用されるため発生します。このロ

ジックでは、Ingress Controller がフロントエンドと見なされ、個々のルートがバックエンドと見なされます。フロントエンド設定に適用されるヘッダー値 **DENY** は、バックエンドで設定されている値 **SAMEORIGIN** で同じヘッダーをオーバーライドします。

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

さらに、Ingress Controller またはルートのいずれかで定義されたアクションは、ルートアノテーションを使用して設定された値をオーバーライドします。

9.1.6.2. 特殊なケースのヘッダー

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

表9.2 特殊な場合のヘッダー設定オプション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
proxy	いいえ	いいえ	**プロキシー** HTTP リクエストヘッダーを使用して、ヘッダー値を **HTTP_PROXY** 環境変数に挿入して、脆弱な CGI アプリケーションを悪用できます。**プロキシー** HTTP リクエストヘッダーも標準ではないため、設定中にエラーが発生しやすくなります。	いいえ
host	いいえ	はい	**IngressController** CR を使用して **ホスト** HTTP 要求ヘッダーが設定されている場合、HAProxy は正しいルートを検索するときに失敗する可能性があります。	いいえ

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

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

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

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

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- プロジェクト管理者として OpenShift Dedicated クラスターにログインしている。

- あるポートを公開する Web アプリケーションと、そのポートでトラフィックをリッスンする HTTP または TCP エンドポイントがある。

手順

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

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

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

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

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

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

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

9.1.8. ルート固有のアノテーション

Ingress Controller 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。Red Hat では、ルートアノテーションの Operator 管理ルートへの追加はサポートしません。

重要

複数の送信元 IP またはサブネットのホワイトリストを作成するには、スペースで区切られたリストを使用します。他の区切りタイプを使用すると、リストが警告やエラーメッセージなしに無視されます。

表9.3 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、**random**、**source**、**roundrobin**、および **leastconn** です。デフォルト値は、TLS パススルールートの場合、**source** です。他のすべてのルートの場合、デフォルトは **random** です。	パススルールートの **ROUTER_TCP_BALANCE_SCHEME** です。それ以外の場合は **ROUTER_LOAD_BALANCE_ALGORITHM** を使用します。
haproxy.router.openshift.io/disable_cookies	関連の接続を追跡する cookie の使用を無効にします。**'true'** または **'TRUE'** に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
router.openshift.io/cookie_name	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	

変数	説明	デフォルトで使用される環境変数
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。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	**'true'** または **'TRUE'** を設定すると、ルートごとに特定のバックエンドの stick-tables で実装されるレート制限機能が有効になります。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	同じ送信元 IP アドレスで行われる同時 TCP 接続の数を制限します。数値を受け入れます。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	同じ送信元 IP アドレスを持つクライアントが HTTP 要求を実行できるレートを制限します。数値を受け入れます。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	同じ送信元 IP アドレスを持つクライアントが TCP 接続を確立するレートを制限します。数値を受け入れます。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
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**

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/timeout-tunnel	このタイムアウトは、クリアテキスト、エッジ、再暗号化、またはパススルーのルートを介した WebSocket などトンネル接続に適用されます。cleartext、edge、または reencrypt のルートタイプでは、このアノテーションは、タイムアウト値がすでに存在するタイムアウトトンネルとして適用されます。パススルーのルートタイプでは、アノテーションは既存のタイムアウト値の設定よりも優先されます。	**ROUTER_DEFAULT_TUNNEL_TIMEOUT**
ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after	設定できるのは、IngressController または Ingress config です。このアノテーションでは、ルーターを再デプロイし、HA プロキシーが haproxy **hard-stop-after** グローバルオプションを実行するように設定します。このオプションは、クリーンなソフトストップ実行で最大許容される時間を定義します。	**ROUTER_HARD_STOP_AFTER**
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	ルートの許可リストを設定します。許可リストは、承認したソースアドレスの IP アドレスおよび CIDR 範囲のリストをスペース区切りにしたリストです。許可リストに含まれていない IP アドレスからの要求は破棄されます。 **haproxy.config** ファイルで直接表示される IP アドレスと CIDR 範囲の最大数は 61 です [1]。	
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strict-Transport-Security ヘッダーを設定します。	
haproxy.router.openshift.io/rewrite-target	バックエンドの要求の書き換えパスを設定します。	

変数	説明	デフォルトで使用される環境変数
router.openshift.io/cookie-same-site	Cookie を制限するために値を設定します。値は以下のようになります。 **Lax**: ブラウザーは、クロスサイト要求では Cookie を送信しませんが、ユーザーが外部サイトから元のサイトに移動するときに Cookie を送信します。これは、**SameSite** 値が指定されていない場合のブラウザーのデフォルトの動作です。 **Strict**: ブラウザーは、同じサイトのリクエストに対してのみ Cookie を送信します。 **None**: ブラウザーは、クロスサイト要求と同一サイト要求の両方に対して Cookie を送信します。 この値は、re-encrypt および edge ルートにのみ適用されます。詳細は、SameSite cookie のドキュメント を参照してください。	
haproxy.router.openshift.io/set-forwarded-headers	ルートごとに **Forwarded** および **X-Forwarded-For** HTTP ヘッダーを処理するポリシーを設定します。値は以下のようになります。 **append**: ヘッダーを追加し、既存のヘッダーを保持します。これはデフォルト値です。 **Replace**: ヘッダーを設定し、既存のヘッダーを削除します。 **never**: ヘッダーを設定しませんが、既存のヘッダーを保持します。 **if-none**: ヘッダーがまだ設定されていない場合にこれを設定します。	**ROUTER_SET_FORWARDED_HEADERS**

1. 許可リストの IP アドレスと CIDR 範囲の数が 61 を超えると、それらは別のファイルに書き込まれます。このファイルは **haproxy.config** から参照されます。このファイルは、**var/lib/haproxy/router/whitelists** フォルダーに保存されます。

注記

アドレスが許可リストに書き込まれることを確認するには、CIDR 範囲の完全なリストが Ingress Controller 設定ファイルに記載されていることを確認します。etcd オブジェクトサイズ制限は、ルートアノテーションのサイズを制限します。このため、許可リストに追加できる IP アドレスと CIDR 範囲の最大数のしきい値が作成されます。

注記

環境変数を編集することはできません。

ルータータイムアウト変数

TimeUnits は数字、その後に単位を指定して表現します。 **us** *(マイクロ秒)、**ms** (ミリ秒、デフォルト)、**s** (秒)、**m** (分)、**h** *(時間)、**d** (日)

正規表現: [1-9][0-9]*(**us**\|**ms**\|**s**\|**m**\|**h**\|**d**)

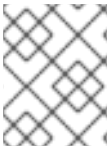
変数	デフォルト	説明
ROUTER_BACKEND_CHECK_INTERVAL	**5000ms**	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	**1s**	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合は、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。
ROUTER_DEFAULT_CLIENT_TIMEOUT	**30s**	クライアントがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_CONNECT_TIMEOUT	**5s**	最大接続時間。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	**1s**	ルーターからルートをバッキングする Pod の TCP FIN タイムアウトを制御します。
ROUTER_DEFAULT_SERVER_TIMEOUT	**30s**	サーバーがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	**1h**	TCP または WebSocket 接続が開放された状態で保つ時間数。このタイムアウト期間は、HAProxy が再読み込みされるたびにリセットされます。

変数	デフォルト	説明
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	**300s**	新しい HTTP 要求が表示されるまで待機する最大時間を設定します。この値が低すぎる場合には、ブラウザーおよびアプリケーションの **keepalive** 値が低くなりすぎて、問題が発生する可能性があります。 有効なタイムアウト値には、想定した個別のタイムアウトではなく、特定の変数を合計した値に指定することができます。たとえば、**ROUTER_SLOWLORIS_HTTP_KEEPALIVE** は、**timeout http-keep-alive** を調整します。HAProxy はデフォルトで **300s** に設定されていますが、HAProxy は **tcp-request inspect-delay** も待機します。これは **5s** に設定されています。この場合、全体的なタイムアウトは **300s** に **5s** を加えたことになります。
ROUTER_SLOWLORIS_TIMEOUT	**10s**	HTTP 要求の伝送にかかる時間。
RELOAD_INTERVAL	**5s**	ルーターがリロードし、新規の変更を受け入れる最小の頻度を許可します。
ROUTER_METRICS_HAPROXY_TIMEOUT	**5s**	HAProxy メトリクスの収集タイムアウト。

ルート設定のカスタムタイムアウト

```
apiVersion: route.openshift.io/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

パススルールートのサーバー側のタイムアウト値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。

特定の IP アドレスを 1 つだけ許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

複数の IP アドレスを許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP アドレスの CIDR ネットワークを許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

IP アドレスと IP アドレスの CIDR ネットワークの両方を許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

書き換えターゲットを指定するルート

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / (1)
...
```

(1) バックエンドの要求の書き換えパスとして / を設定します。

ルートに **haproxy.router.openshift.io/rewrite-target** アノテーションを設定すると、要求をバックエンドアプリケーションに転送する前に Ingress Controller がこのルートを使用して HTTP 要求のパスを書き換える必要があることを指定します。**spec.path** で指定されたパスに一致する要求パスの一部は、アノテーションで指定された書き換えターゲットに置き換えられます。

以下の表は、**spec.path**、要求パス、および書き換えターゲットの各種の組み合わせに関するパスの書き換え動作の例を示しています。

表9.4 rewrite-target の例

Route.spec.path	要求パス	書き換えターゲット	転送された要求パス
/foo	/foo	/	/
/foo	/foo/	/	/
/foo	/foo/bar	/	/bar

Route.spec.path	要求パス	書き換えターゲット	転送された要求パス
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/
/foo	/foo/bar	/baz	/baz/bar
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	該当なし (要求パスがルートパスに一致しない)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

haproxy.router.openshift.io/rewrite-target 内の特定の特殊文字は、適切にエスケープする必要があるため、特別な処理が必要です。これらの文字がどのように処理されるかについては、次の表を参照してください。

表9.5 特殊文字の取り扱い

以下の文字の場合	以下の文字を使用	注記
#	\#	# は書き換え式を終了させるので使用しないでください。
%	% または %%	%%% のような変則的なシーケンスは避けてください。
‘	\’	‘ は無視されるので避けてください。

その他の有効な URL 文字はすべてエスケープせずに使用できます。

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

TLS 設定を指定せずに Ingress オブジェクトを作成すると、OpenShift Dedicated は安全でないルートを生成します。デフォルトの Ingress 証明書を使用してセキュアなエッジ終端ルートを生成する Ingress オブジェクトを作成するには、次のように空の TLS 設定を指定できます。

前提条件

- 公開したいサービスがあります。

- OpenShift CLI (**oc**) にアクセスできる。

手順

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

   Ingress オブジェクトの YAML 定義

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

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

2. 次のコマンドを実行して、Ingress オブジェクトを作成します。

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

検証

- 以下のコマンドを実行して、OpenShift Dedicated が Ingress オブジェクトの予期されるルートを作成したことを確認します。

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

  出力例

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

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

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

3 ルートは、**edge** の終了ポリシーを指定する必要があります。

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

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

前提条件

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

手順

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

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

1 アノテーションは kubernetes シークレットを参照します。

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

出力例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
```

```
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

関連情報

- appsDomain オプションを使用した代替クラスタードメインの指定

9.2. セキュリティー保護されたルート

セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。以下のセクションでは、カスタム証明書を使用して re-encrypt、edge、および passthrough ルートを作成する方法を説明します。

重要

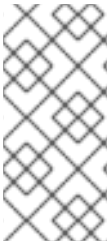
パブリックエンドポイントを使用して Microsoft Azure にルートを作成する場合、リソース名は制限されます。特定の用語を使用するリソースを作成することはできません。Azure が制限する語のリストは、Azure ドキュメントの Resolve reserved resource name errors を参照してください。

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



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 Controller 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: route.openshift.io/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 Controller 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 Controller 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/キーのペアが必要です。ここで、証明書はルートホストに対して有効となっています。

- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- 公開する必要のあるサービスが必要です。

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サービスの名前に置き換えます。**www.example.com** を適切な名前に置き換えます。

- edge TLS termination およびカスタム証明書を使用して、セキュアな **Route** リソースを作成します。

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

結果として生成される **Route** リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
apiVersion: route.openshift.io/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。

9.2.3. passthrough ルートの作成

oc create route コマンドを使用し、passthrough termination を使用してセキュアなルートを設定できます。passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、ルートでキーや証明書は必要ありません。

前提条件

- 公開する必要のあるサービスが必要です。

手順

- **Route** リソースを作成します。

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

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。

passthrough termination を使用したセキュリティー保護されたルート

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough (2)
    insecureEdgeTerminationPolicy: None (3)
  to:
    kind: Service
    name: frontend
```

(1) オブジェクトの名前で、63 文字に制限されます。

(2) **termination** フィールドを **passthrough** に設定します。これは、必要な唯一の **tls** フィールドです。

(3) オプションの **insecureEdgeTerminationPolicy**。唯一有効な値は **None**、**Redirect**、または空の値です (無効にする場合)。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。

9.2.4. 外部管理証明書を使用したルートの作成

重要

TLS シークレット内の外部証明書を使用してルートを保護する機能は、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 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

ルート API の **.spec.tls.externalCertificate** フィールドを使用して、サードパーティーの証明書管理ソリューションで OpenShift Dedicated ルートを設定できます。シークレットを介して外部で管理されている TLS 証明書を参照できるため、手動での証明書管理が不要になります。外部で管理される証明書を使用するとエラーが減り、証明書の更新がよりスムーズに展開されるため、OpenShift ルーターは更新された証明書を迅速に提供できるようになります。

注記

この機能は、エッジルートと再暗号化ルートの両方に適用されます。

前提条件

- **RouteExternalCertificate** フィーチャーゲートを有効にする必要があります。
- **routes/custom-host** に対する **create** および **update** 権限が必要です。
- **tls.key** キーと **tls.crt** キーの両方を含む、**kubernetes.io/tls** タイプの PEM エンコード形式の有効な証明書/キーペアを含むシークレットが必要です。
- 参照されるシークレットは、保護するルートと同じ namespace に配置する必要があります。

手順

1. 次のコマンドを実行して、シークレットと同じ namespace に **role** を作成し、ルーターサービスアカウントに読み取りアクセスを許可します。

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-name=
<secret-name> \ 1
--namespace=<current-namespace> 2
```

1. シークレットの実際の名前を指定します。
2. シークレットとルートの両方が存在する namespace を指定します。

2. 次のコマンドを実行して、シークレットと同じ namespace に **rolebinding** を作成し、ルーターサービスアカウントを新しく作成されたロールにバインドします。

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace> 1
```

1. シークレットとルートの両方が存在する namespace を指定します。

3. 次の例を使用して、**route** を定義し、証明書を含むシークレットを指定する YAML ファイルを作成します。

セキュアなルートの YAML 定義

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: myedge
  namespace: test
spec:
  host: myedge-test.apps.example.com
  tls:
    externalCertificate:
      name: <secret-name> 1
    termination: edge
    [...]
[...]
```

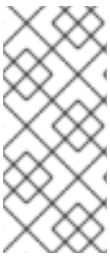
1 シークレットの実際の名前を指定します。

4. 次のコマンドを実行して **route** リソースを作成します。

```
$ oc apply -f <route.yaml> 1
```

1 生成された YAML ファイル名を指定します。

シークレットが存在し、証明書/キーペアがある場合、すべての前提条件が満たされていれば、ルーターは生成された証明書を提供します。



注記

.spec.tls.externalCertificate が指定されていないと、ルーターはデフォルトで生成された証明書を使用します。

.spec.tls.externalCertificate フィールドを使用する場合は、**.spec.tls.certificate** フィールドまたは **.spec.tls.key** フィールドを指定することはできません。

関連情報

- 外部で管理される証明書を使用したルートのトラブルシューティングについては、OpenShift Dedicated ルーター Pod のログでエラーを確認してください。Pod の問題の調査 を参照してください。