OpenShift Dedicated 4

网络

配置 OpenShift Dedicated 网络

Last Updated: 2022-11-28

OpenShift Dedicated 4 网络

配置 OpenShift Dedicated 网络

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本文档提供有关 OpenShift Dedicated 集群联网的信息。

目录

第 1 章 OPENSHIFT DEDICATED 中的 INGRESS OPERATOR

1.1. OPENSHIFT DEDICATED INGRESS OPERATOR

当您创建 OpenShift Dedicated 集群时，集群中运行的 pod 和服务会为每个分配自己的 IP 地址。IP 地址可供附近运行的其他容器集和服务访问，但外部客户端无法访问这些 IP 地址。Ingress Operator 实施 **IngressController** API，它是负责启用对 OpenShift Dedicated 集群服务的外部访问的组件。

Ingress Operator 通过部署和管理一个或多个基于 HAProxy 的 Ingress Controller 来处理路由，使外部客户端可以访问您的服务。Red Hat Site Reliabilitys(SRE)管理 OpenShift Dedicated 集群的 Ingress Operator。虽然您无法更改 Ingress Operator 的设置，但您可以查看默认的 Ingress Controller 配置、状态和日志以及 Ingress Operator 状态。

1.2. 查看默认的 INGRESS CONTROLLER

Ingress Operator 是 OpenShift Dedicated 的一个核心功能，开箱即用。

每个新的 OpenShift Dedicated 安装都有一个名为 default 的 **ingresscontroller**。它可以通过额外的 Ingress Controller 来补充。如果删除了默认的 **ingresscontroller**，Ingress Operator 会在一分钟内自动重新创建。

流程

- 查看默认的 Ingress Controller：

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

1.3. 查看 INGRESS OPERATOR 状态

您可以查看并检查 Ingress Operator 的状态。

流程

- 查看您的 Ingress Operator 状态：

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

1.4. 查看 INGRESS CONTROLLER 日志

您可以查看 Ingress Controller 日志。

流程

- 查看 Ingress Controller 日志：

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

1.5. 查看 INGRESS CONTROLLER 状态

您可以查看特定 Ingress Controller 的状态。

流程

- 查看 Ingress Controller 的状态：

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

1.6. OPENSHIFT DEDICATED INGRESS OPERATOR 配置

下表详细介绍了 Ingress Operator 的组件，以及 Red Hat Site Reliabilitys(SRE)在 OpenShift Dedicated 集群上维护这个组件。

表 1.1. Ingress Operator Responsibility Chart

Ingress 组件	受管理	默认配置？
扩展 Ingress Controller	SRE	是
Ingress Operator 线程计数	SRE	是
Ingress Controller 访问日志	SRE	是
Ingress Controller 分片	SRE	是
Ingress Controller 路由准入策略	SRE	是
Ingress Controller 通配符路由	SRE	是
Ingress Controller X-Forwarded 标头	SRE	是
Ingress Controller 路由压缩	SRE	是

第 2 章 OPENSHIFT SDN 默认 CNI 网络供应商

2.1. 为项目启用多播

2.1.1. 关于多播

通过使用 IP 多播，数据可同时广播到许多 IP 地址。

重要

目前，多播最适用于低带宽协调或服务发现。它不是一个高带宽解决方案。

默认情况下，OpenShift Dedicated pod 之间多播流量被禁用。如果使用 OpenShift SDN 默认 Container Network Interface (CNI) 网络供应商，可以根据每个项目启用多播。

以 **networkpolicy** 隔离模式使用 OpenShift SDN 网络插件时：

- pod 发送的多播数据包将传送到项目中的所有其他 pod，而无需考虑 **NetworkPolicy** 对象。即使在无法通过单播通信时，Pod 也能通过多播进行通信。
- 一个项目中的 pod 发送的多播数据包不会传送到任何其他项目中的 pod，即使存在允许项目间通信的 **NetworkPolicy** 对象。

以 **multitenant** 隔离模式使用 OpenShift SDN 网络插件时：

- pod 发送的多播数据包将传送到项目中的所有其他 pod。
- 只有在各个项目接合在一起并且每个接合的项目上都启用了多播时，一个项目中的 pod 发送的多播数据包才会传送到其他项目中的 pod。

2.1.2. 启用 pod 间多播

您可以为项目启用 pod 间多播。

先决条件

- 安装 OpenShift CLI(**oc**)。
- 您必须使用具有 **cluster-admin** 或 **dedicated-admin** 角色的用户登录到集群。

流程

- 运行以下命令，为项目启用多播。使用您要启用多播的项目的名称替换 **<namespace>**。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

验证

要验证项目是否启用了多播，请完成以下步骤：

1. 将您的当前项目更改为启用多播的项目。使用项目名替换 **<project>**。

```
$ oc project <project>
```

2. 创建 pod 以作为多播接收器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. 创建 pod 以作为多播发送器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 在新的终端窗口或选项卡中，启动多播监听程序。

   a. 获得 Pod 的 IP 地址：

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. 输入以下命令启动多播监听程序：

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. 启动多播传输。

   a. 获取 pod 网络 IP 地址范围：

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

   b. 要发送多播信息，请输入以下命令：

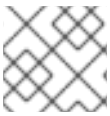
   ```
   $ oc exec msender -i -t -- \
       /bin/bash -c "echo | socat STDIO UDP4-
   DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
   ```

   如果多播正在工作，则上一个命令会返回以下输出：

   ```
   mlistener
   ```

第 3 章 配置集群范围代理

如果您使用现有的 Virtual Private Cloud (VPC)，您可以在 OpenShift Dedicated 集群安装过程中或安装集群后配置集群范围的代理。当您启用代理时，核心集群组件会被拒绝访问互联网，但代理不会影响用户工作负载。



注意

只有集群系统出口流量会被代理，包括对云供应商 API 的调用。

您只能对使用客户 Cloud Subscription (CCS)模型的 OpenShift Dedicated 集群启用代理。

如果使用集群范围代理，您需要维护到集群的代理可用性。如果代理不可用，这可能会影响集群的健康和支持性。

3.1. 配置集群范围代理的先决条件

要配置集群范围代理，您必须满足以下要求。当您在安装过程中或安装后配置代理时，这些要求有效。

常规要求

- 您是集群所有者。
- 您的帐户有足够的权限。
- 已为集群有一个现有的 Virtual Private Cloud (VPC)。
- 您为集群使用客户云订阅(CCS)模式。
- 代理可以访问集群的 VPC 和 VPC 的专用子网。此代理也可以从 VPC 的集群以及 VPC 的专用子网访问。
- 您以将 **ec2.<region>.amazonaws.com**, **elasticloadbalancing.<region>.amazonaws.com**, 和 **s3.<region>.amazonaws.com** 端点添加到您的 VPC 端点。需要这些端点才能完成节点到 AWS EC2 API 的请求。由于代理在容器级别工作，而不是在节点级别，因此您必须通过 AWS 专用网络将这些请求路由到 AWS EC2 API。在您的代理服务器中的允许列表中添加 EC2 API 的公共 IP 地址是不够的。

网络要求

- 如果您的代理重新排序出口流量，您必须创建指向域和端口组合的排除项。下表针对这些例外提供了指导。
  - 将以下 OpenShift URL 添加到 allowlist 中用于重新加密。

address	协议/门户	功能
observatorium-mst.api.openshift.com	https/443	必需。用于受管 OpenShift 特定的遥测。

address	协议/门户	功能
sso.redhat.com	https/443	https://cloud.redhat.com/openshift 站点使用 sso.redhat.com 上的身份验证来下载集群 pull secret，并使用 Red Hat SaaS 解决方案来帮助监控您的订阅、集群清单和计费报告。

- 在允许列表中添加以下站点可靠性工程(SRE)和管理 URL 来重新加密。

address	协议/门户	功能
***.osdsecuritylogs.splunkcloud.com** **或者** **inputs1.osdsecuritylogs.splunkcloud.com** **inputs2.osdsecuritylogs.splunkcloud.com** **inputs4.osdsecuritylogs.splunkcloud.com** **inputs5.osdsecuritylogs.splunkcloud.com** **inputs6.osdsecuritylogs.splunkcloud.com** **inputs7.osdsecuritylogs.splunkcloud.com** **inputs8.osdsecuritylogs.splunkcloud.com** **inputs9.osdsecuritylogs.splunkcloud.com** **inputs10.osdsecuritylogs.splunkcloud.com** **inputs11.osdsecuritylogs.splunkcloud.com** **inputs12.osdsecuritylogs.splunkcloud.com** **inputs13.osdsecuritylogs.splunkcloud.com 输入** **14.osdsecuritylogs.splunkcloud.com** **inputs15.osdsecuritylogs.splunkcloud.com**	tcp/9997	由 splunk-forwarder-operator 使用，作为 Red Hat SRE 用于基于日志的警报的日志转发端点。

address	协议/门户	功能
http-inputs-osdsecuritylogs.splunkcloud.com	https/443	由 splunk-forwarder-operator 使用，作为 Red Hat SRE 用于基于日志的警报的日志转发端点。

重要

目前，如果服务器充当透明的转发代理（其中没有通过 **--http-proxy** 或 **--https-proxy** 参数），则不支持使用代理服务器来执行 TLS 重新加密。

透明的转发代理会截获集群流量，但它实际上并没有在集群本身配置。

其它资源

- 有关使用客户云订阅(CCS)模型的 OpenShift Dedicated 集群的安装先决条件，请参阅 AWS 或 GCP 上的 客户云订阅。

3.2. 其他信任捆绑包的职责

如果您提供额外的信任捆绑包，您需要进行以下要求：

- 确保其他信任捆绑包的内容有效
- 确保证书（包括中间证书）包含在额外的信任捆绑包中
- 跟踪到期，并为附加信任捆绑包中包含的证书执行必要的续订
- 使用更新的额外信任捆绑包更新集群配置

3.3. 在安装过程中配置代理

在使用客户云订阅(CCS)集群安装到现有的 Virtual Private Cloud (VPC)集群时，可以配置 HTTP 或 HTTPS 代理。您可以使用 Red Hat OpenShift Cluster Manager 在安装过程中配置代理。

3.4. 使用 OPENSHIFT CLUSTER MANAGER 在安装过程中配置代理

如果您要将 OpenShift Dedicated 集群安装到现有的 Virtual Private Cloud (VPC)，您可以使用 Red Hat OpenShift Cluster Manager 在安装过程中启用集群范围的 HTTP 或 HTTPS 代理。您只能针对使用客户 Cloud Subscription (CCS)模型的集群启用代理。

在安装前，您必须验证可以从 VPC 访问代理，该代理是否可从安装到的 VPC 中。该代理还必须从 VPC 的专用子网访问。

有关使用 OpenShift Cluster Manager 在安装过程中配置集群范围代理的详细信息，请参阅使用 CCS *在 AWS 上创建集群*，或使用 CCS *在 GCP 上创建集群*。

其它资源

- 使用 CCS 在 AWS 上创建集群
- 使用 CCS 在 GCP 上创建集群

3.5. 安装后配置代理

在使用客户云订阅(CCS)集群安装到现有的 Virtual Private Cloud (VPC)后，可以配置 HTTP 或 HTTPS 代理。您可以使用 Red Hat OpenShift Cluster Manager 安装后配置代理。

3.6. 使用 OPENSHIFT CLUSTER MANAGER 在安装后配置代理

您可以使用 Red Hat OpenShift Cluster Manager 在 Virtual Private Cloud (VPC)中的现有 OpenShift Dedicated 集群中添加集群范围的代理配置。您只能针对使用客户 Cloud Subscription (CCS)模型的集群启用代理。

您还可以使用 OpenShift Cluster Manager 更新现有的集群范围代理配置。例如，如果代理的任何证书颁发机构过期，您可能需要更新代理的网络地址，或者替换额外的信任捆绑包。

重要

集群将代理配置应用到 control plane 和计算节点。在应用配置时，每个集群节点暂时处于不可调度状态，并排空其工作负载。每个节点都会作为进程的一部分重启。

先决条件

- 您有一个 OpenShift Dedicated 集群，它使用 Customer Cloud Subscription (CCS)模型。
- 您的集群部署在 VPC 中。

流程

1. 导航到 OpenShift Cluster Manager Hybrid Cloud Console 并选择您的集群。
2. 在 **Networking** 页面上的 **Virtual Private Cloud (VPC)**部分下，点 **Edit cluster-wide proxy**。
3. 在 **Edit cluster-wide proxy** 页面中，提供代理配置详情：
   a. 至少在以下字段之一中输入值：
      - 指定有效的 **HTTP 代理 URL**。
      - 指定有效的 **HTTPS 代理 URL**。
      - 在 **Additional trust bundle** 字段中，提供 PEM 编码 X.509 证书捆绑包。如果您要替换现有的信任捆绑包文件，请选择 **replace file** 来查看字段。捆绑包添加到集群节点的可信证书存储中。需要额外的信任捆绑包文件，除非代理的身份证书由 Red Hat Enterprise Linux CoreOS (RHCOS)信任捆绑包的颁发机构签名。
        如果您使用不要求额外代理配置但需要额外的证书颁发机构(CA)的 MITM 透明代理网络，您必须提供 MITM CA 证书。

注意

如果您在没有指定 HTTP 或 HTTPS 代理 URL 的情况下上传额外的信任捆绑包文件，则会在集群中设置捆绑包，但不会配置为与代理一起使用。

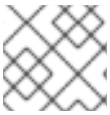
b. 单击 **Confirm**。

验证

- 在 **Networking** 页面上的 **Virtual Private Cloud (VPC)**部分下，验证集群的代理配置是否如预期。

第 4 章 CIDR 范围定义

您必须为以下 CIDR 范围指定非重叠范围。



注意

创建集群后无法更改机器 CIDR 范围。

4.1. MACHINE CIDR

在 Machine CIDR 字段中，必须为机器或集群节点指定 IP 地址范围。这个范围必须包括虚拟私有云(VPC)子网的所有 CIDR 地址范围。子网必须相邻。单个可用区部署的 IP 地址范围为 128 个地址，使用子网前缀 **/25**。对于使用多个可用区的部署，支持使用子网前缀 **/24** 的最小地址范围。默认值为 **10.0.0.0/16**。这个范围不得与任何连接的网络冲突。

4.2. SERVICE CIDR

在 Service CIDR 字段中，必须为服务指定 IP 地址范围。范围必须足够大，以适应您的工作负载。该地址块不得与从集群内部访问的任何外部服务重叠。默认为 **172.30.0.0/16**。这个地址块需要在集群间相同。

4.3. POD CIDR

在 pod CIDR 字段中，必须为 pod 指定 IP 地址范围。范围必须足够大，以适应您的工作负载。该地址块不得与从集群内部访问的任何外部服务重叠。默认为 **10.128.0.0/14**。这个地址块需要在集群间相同。

4.4. 主机前缀

在 Host Prefix 字段中，您必须指定分配给分配给各个机器的 pod 的子网前缀长度。主机前缀决定了每台机器的 pod IP 地址池。例如，如果主机前缀设置为 **/23**，则每台机器从 pod CIDR 地址范围分配一个 **/23** 子网。默认值为 **/23**，允许每个节点 512 个集群节点和 512 个 pod（这两部分都受到支持）。

第 5 章 网络策略

5.1. 关于网络策略

作为集群管理员，您可以定义网络策略以限制到集群中的 pod 的网络通讯。

5.1.1. 关于网络策略

在使用支持 Kubernetes 网络策略的 Kubernetes Container Network Interface（CNI）插件的集群中，网络隔离完全由 **NetworkPolicy** 对象控制。在 OpenShift Dedicated 4 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。

警告

网络策略不适用于主机网络命名空间。启用主机网络的 Pod 不受网络策略规则的影响。

默认情况下，项目中的所有 pod 都可被其他 pod 和网络端点访问。要在一个项目中隔离一个或多个 Pod，您可以在该项目中创建 **NetworkPolicy** 对象来指示允许的入站连接。项目管理员可以在自己的项目中创建和删除 **NetworkPolicy** 对象。

如果一个 pod 由一个或多个 **NetworkPolicy** 对象中的选择器匹配，那么该 pod 将只接受至少被其中一个 **NetworkPolicy** 对象所允许的连接。未被任何 **NetworkPolicy** 对象选择的 pod 可以完全访问。

以下示例 **NetworkPolicy** 对象演示了支持不同的情景：

- 拒绝所有流量：
  要使项目默认为拒绝流量，请添加一个匹配所有 pod 但不接受任何流量的 **NetworkPolicy** 对象：

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

- 只允许 OpenShift Dedicated Ingress Controller 的连接：
  要使项目只允许 OpenShift Dedicated Ingress Controller 的连接，请添加以下 **NetworkPolicy** 对象。

```
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
```

- 只接受项目中 pod 的连接：
  要使 pod 接受同一项目中其他 pod 的连接，但拒绝其他项目中所有 pod 的连接，请添加以下 **NetworkPolicy** 对象：

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

- 仅允许基于 pod 标签的 HTTP 和 HTTPS 流量：
  要对带有特定标签（以下示例中的 **role=frontend**）的 pod 仅启用 HTTP 和 HTTPS 访问，请添加类似如下的 **NetworkPolicy** 对象：

```
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
```

- 使用命名空间和 pod 选择器接受连接：
  要通过组合使用命名空间和 pod 选择器来匹配网络流量,您可以使用类似如下的 **NetworkPolicy** 对象：

```
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 对象是可添加的；也就是说，您可以组合多个 **NetworkPolicy** 对象来满足复杂的网络要求。

例如，对于以上示例中定义的 **NetworkPolicy** 对象，您可以在同一个项目中定义 **allow-same-namespace** 和 **allow-http-and-https** 策略。因此，允许带有标签 **role=frontend** 的 pod 接受每一策略所允许的任何连接。即，任何端口上来自同一命名空间中的 pod 的连接，以及端口 **80** 和 **443** 上的来自任意命名空间中 pod 的连接。

5.1.2. 网络策略优化

使用一个网络策略来通过 pod 上的不同标签来在命名空间中将不同 pod 进行隔离。

注意

有效使用网络策略规则的指南只适用于 OpenShift SDN 集群网络供应商。

将 **NetworkPolicy** 对象应用到单一命名空间中的大量 pod 时，效率较低。因为 Pod 标签不存在于 IP 地址一级，因此网络策略会为使用 **podSelector** 选择的每个 pod 之间生成单独的 Open vSwitch（OVS）流量规则 。

例如，在一个 **NetworkPolicy** 对象中，如果 spec **podSelector** 和 ingress **podSelector** 每个都匹配 200 个 pod，则会产生 40,000 (200*200) OVS 流规则。这可能会减慢节点的速度。

在设计您的网络策略时，请参考以下指南：

- 使用命名空间使其包含需要隔离的 pod 组，可以减少 OVS 流规则数量。
  使用 **namespaceSelector** 或空 **podSelector** 选择整个命名空间的**NetworkPolicy** 对象会只生成一个与命名空间的 VXLAN 虚拟网络 ID（VNID）匹配的 OVS 流量规则。
- 保留不需要在原始命名空间中隔离的 pod，并将需要隔离的 pod 移到一个或多个不同的命名空间中。
- 创建额外的目标跨命名空间网络策略，以允许来自不同隔离的 pod 的特定流量。

5.1.3. 后续步骤

- 创建网络策略

5.2. 创建网络策略

作为具有 **admin** 角色的用户，您可以为命名空间创建网络策略。

5.2.1. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
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 对象的名称。
2. 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。
3. 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。
4. 接受流量的一个或多个目标端口的列表。

5.2.2. 使用 CLI 创建网络策略

要定义细致的规则来描述集群中命名空间允许的入口或出口网络流量，您可以创建一个网络策略。

注意

如果使用具有 **cluster-admin** 角色的用户登录，则可以在集群中的任何命名空间中创建网络策略。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略规则：
   a. 创建一个 **<policy_name>.yaml** 文件：

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

   其中：

<policy_name>

指定网络策略文件名。

b. 在您刚才创建的文件中定义网络策略，如下例所示：

拒绝来自所有命名空间中的所有 pod 的入口流量

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

允许来自所有命名空间中的所有 pod 的入口流量

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

2. 运行以下命令来创建网络策略对象：

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

其中：

<policy_name>

指定网络策略文件名。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

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

注意

如果您使用 **cluster-admin** 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式创建网络策略。

5.2.3. 使用 OpenShift Cluster Manager 创建网络策略

要定义细致的规则来描述集群中命名空间允许的入口或出口网络流量，您可以创建一个网络策略。

先决条件

- 登录 OpenShift Cluster Manager 混合云控制台。
- 已创建一个 OpenShift Dedicated 集群。
- 已为集群配置身份提供程序。
- 将您的用户帐户添加到配置的身份提供程序中。
- 您在 OpenShift Dedicated 集群中创建了项目。

流程

1. 在 OpenShift Cluster Manager Hybrid Cloud Console 中，点您要访问的集群。
2. 单击 **Open console** 以导航到 OpenShift Web 控制台。
3. 点击身份提供程序，并提供您的凭证以登录到集群。
4. 从管理员视角，在 **Networking** 下点击 **NetworkPolicies**。
5. 点 **Create NetworkPolicy**。
6. 在 **Policy name** 字段中，提供策略的名称。
7. 可选：如果此策略仅适用于一个或多个特定的 pod，您可以为特定 pod 提供标签和选择器。如果您没有选择特定 pod，则此策略将适用于集群中的所有 pod。
8. 可选： 您可以使用 **Deny 所有入口流量或拒绝所有 出口流量复选框来阻止所有入口和出口流量**。
9. 您还可以添加入口和出口规则的任意组合，允许您指定您要批准的端口、命名空间或 IP 块。
10. 在您的策略中添加入站规则：
    a. 选择 **Add ingress 规则** 来配置新规则。此操作使用 **Add allowed source** 下拉菜单创建新的 **Ingress 规则** 行，允许您指定如何限制入站流量。下拉菜单提供三个选项来限制您的入口流量：
       - **允许来自同一命名空间的** pod 将流量限制到同一命名空间中的 pod。您可以在命名空间中指定 pod，但将此选项留空允许来自该命名空间中的所有 pod 的流量。
       - **允许从集群内部的 pod** 将流量限制到与策略相同的集群中的 pod。您可以指定要允许入站流量的命名空间和 pod。将此选项留空可让来自此集群中所有命名空间和 pod 的入站流量。
       - **允许 IP** 块限制指定无域间路由(CIDR)IP 块的流量。您可以使用例外选项阻止特定的 IP。将 CIDR 字段留空允许所有外部来源的所有入站流量。
    b. 您可以将所有入站流量限制为端口。如果您不添加任何端口，则流量可以访问所有端口。
11. 在您的网络策略中添加出口规则：
    a. 选择 **Add egress rule** 来配置新规则。此操作会创建一个带有 **Add allowed destination**"* 下拉菜单的新 **Egress 规则** 行，它允许您指定如何限制出站流量。下拉菜单提供三个选项来限制您的出口流量：
       - **允许来自同一命名空间的 pod** 将出站流量限制为同一命名空间中的 pod。您可以在命名空间中指定 pod，但将此选项留空允许来自该命名空间中的所有 pod 的流量。

- **允许从集群内部的 pod** 将流量限制到与策略相同的集群中的 pod。您可以指定要允许出站流量的命名空间和 pod。将这个选项留空允许来自此集群中所有命名空间和 pod 的出站流量。
- **允许 IP** 块限制指定 CIDR IP 块的流量。您可以使用例外选项阻止特定的 IP。将 CIDR 字段留空允许所有外部来源的出站流量。

b. 您可以将所有出站流量限制为端口。如果您不添加任何端口，则流量可以访问所有端口。

5.3. 查看网络策略

以具有 **admin** 角色的用户，您可以查看命名空间的网络策略。

5.3.1. 示例 NetworkPolicy 对象

下文解释了示例 NetworkPolicy 对象：

```
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 对象的名称。
2. 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。
3. 与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。
4. 接受流量的一个或多个目标端口的列表。

5.3.2. 使用 CLI 查看网络策略

您可以检查命名空间中的网络策略。

注意

如果使用具有 **cluster-admin** 角色的用户登录，您可以查看集群中的任何网络策略。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

- 列出命名空间中的网络策略：
  - 要查看命名空间中定义的网络策略对象，请输入以下命令：

    ```
    $ oc get networkpolicy
    ```

  - 可选： 要检查特定的网络策略，请输入以下命令：

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

    其中：

    <policy_name>
    指定要检查的网络策略的名称。

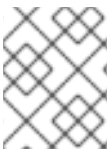
    <namespace>
    可选： 如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

    例如：

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

    oc describe 命令的输出

    ```
    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
    ```



注意

如果您使用 **cluster-admin** 权限登录到 web 控制台，您可以选择在集群中的任何命名空间中以 YAML 或 web 控制台的形式查看网络策略。

5.3.3. 使用 OpenShift Cluster Manager 查看网络策略

您可以在 Red Hat OpenShift Cluster Manager 中查看网络策略的配置详情。

先决条件

- 登录 OpenShift Cluster Manager 混合云控制台。
- 已创建一个 OpenShift Dedicated 集群。
- 已为集群配置身份提供程序。
- 将您的用户帐户添加到配置的身份提供程序中。
- 您创建了网络策略。

流程

1. 从 OpenShift Cluster Manager Web 控制台的 **Administrator** 视角，在 **Networking** 下点击 **NetworkPolicies**。
2. 选择要查看的网络策略。
3. 在 **Network Policy** 详情页面中，您可以查看所有相关入口和出口规则。
4. 选择网络策略详情上的 **YAML** 以 YAML 格式查看策略配置。

注意

您只能查看这些策略的详情。您不能编辑这些策略。

5.4. 删除网络策略

以具有 **admin** 角色的用户，您可以从命名空间中删除网络策略。

5.4.1. 使用 CLI 删除网络策略

您可以删除命名空间中的网络策略。

注意

如果使用具有 **cluster-admin** 角色的用户登录，您可以删除集群中的任何网络策略。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

- 要删除网络策略对象，请输入以下命令：

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

其中：

<policy_name>

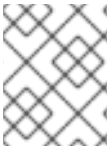
指定网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

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



注意

如果使用 **cluster-admin** 权限登录到 web 控制台，您可以选择在集群上以 YAML 或通过 **Actions** 菜单从 web 控制台中的策略删除网络策略。

5.4.2. 使用 OpenShift Cluster Manager 删除网络策略

您可以删除命名空间中的网络策略。

先决条件

- 登录 OpenShift Cluster Manager 混合云控制台。
- 已创建一个 OpenShift Dedicated 集群。
- 已为集群配置身份提供程序。
- 将您的用户帐户添加到配置的身份提供程序中。

流程

1. 从 OpenShift Cluster Manager Web 控制台的 **Administrator** 视角，在 **Networking** 下点击 **NetworkPolicies**。
2. 使用以下方法删除您的网络策略：
   - 从 **Network Policies** 表中删除策略：
     a. 在 **Network Policies** 表中，选择您要删除的网络策略行的堆栈菜单，然后点击 **Delete NetworkPolicy**。
   - 使用独立网络策略详情中的 **Actions** 下拉菜单删除策略：
     a. 点击网络策略的 **Actions** 下拉菜单。
     b. 从菜单中选择 **Delete NetworkPolicy**。

5.5. 使用网络策略配置多租户隔离

作为集群管理员，您可以配置网络策略以为多租户网络提供隔离功能。

注意

如果使用 OpenShift SDN 集群网络供应商，请按照本节所述配置网络策略，提供类似于多租户模式的网络隔离，但具有设置网络策略模式。

5.5.1. 使用网络策略配置多租户隔离

您可以配置项目，使其与其他项目命名空间中的 pod 和服务分离。

先决条件

- 集群使用支持 **NetworkPolicy** 对象的集群网络供应商，如设置了 **mode: NetworkPolicy** 的 OpenShift SDN 网络供应商。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（**oc**）。
- 您可以使用具有 **admin** 权限的用户登陆到集群。

流程

1. 创建以下 **NetworkPolicy** 对象：
   a. 名为 **allow-from-openshift-ingress** 的策略。

```
$ 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
```

注意

policy-group.network.openshift.io/ingress: "" 是 OpenShift SDN 的首选命名空间选择器标签。您可以使用 **network.openshift.io/policy-group: ingress** 命名空间选择器标签，但这是一个比较旧的用法。

   b. 名为 **allow-from-openshift-monitoring** 的策略：

```
$ 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. 名为 **allow-same-namespace** 的策略：

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

d. 名为 **allow-from-kube-apiserver-operator** 的策略：

```
$ 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
```

如需了解更多详细信息，请参阅 New **kube-apiserver-operator** Webhook 控制器验证 webhook 的健康状况。

2. 可选：要确认当前项目中存在网络策略，请输入以下命令：

```
$ oc describe networkpolicy
```

输出示例

```
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 6 章 配置路由

6.1. 路由配置

6.1.1. 创建基于 HTTP 的路由

路由允许您在公共 URL 托管应用程序。根据应用程序的网络安全配置，它可以安全或不受保护。基于 HTTP 的路由是一个不受保护的路由，它使用基本的 HTTP 路由协议，并在未安全的应用程序端口上公开服务。

以下流程描述了如何使用 **hello-openshift** 应用程序创建基于 HTTP 的简单路由，作为示例。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 以管理员身份登录。
- 您有一个 web 应用，用于公开端口和侦听端口上流量的 TCP 端点。

流程

1. 运行以下命令，创建一个名为 **hello-openshift** 的项目：

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

2. 运行以下命令，在项目中创建 pod：

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

3. 运行以下命令，创建名为 **hello-openshift** 的服务：

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

4. 运行以下命令，创建一个没有安全安全的路由到 **hello-openshift** 应用程序：

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

验证

- 要验证您创建的 **路由资源**，请运行以下命令：

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

1 在本例中，路由名为 **hello-openshift**。

创建的未安全路由的 YAML 定义示例：

```
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>** 是默认的入口域名。**ingresses.config/cluster** 对象在安装过程中创建，且无法更改。如果要指定不同的域，您可以使用 **appsDomain** 选项指定备用集群域。

2 **targetPort** 是此路由所指向的服务选择的 pod 上的目标端口。

注意

要显示您的默认入口域，请运行以下命令：

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

6.1.2. 配置路由超时

如果您的服务需要低超时（满足服务级别可用性 (SLA) 目的）或高超时（具有慢速后端的情况），您可以为现有路由配置默认超时。

先决条件

- 您需要在运行的集群中部署了 Ingress Controller。

流程

1. 使用 **oc annotate** 命令，为路由添加超时：

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1 支持的时间单位是微秒 (us)、毫秒 (ms)、秒钟 (s)、分钟 (m)、小时 (h)、或天 (d)。

以下示例在名为 **myroute** 的路由上设置两秒的超时：

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

6.1.3. HTTP 严格传输安全性

HTTP 严格传输安全性 (HSTS) 策略是一种安全增强，向浏览器客户端发送信号，表示路由主机上仅允许 HTTPS 流量。HSTS 也通过信号 HTTPS 传输来优化 Web 流量，无需使用 HTTP 重定向。HSTS 对于加快与网站的交互非常有用。

强制 HSTS 策略时，HSTS 会向站点的 HTTP 和 HTTPS 响应添加 Strict Transport Security 标头。您可以在路由中使用 **insecureEdgeTerminationPolicy** 值，以将 HTTP 重定向到 HTTPS。强制 HSTS 时，客户端会在发送请求前将所有请求从 HTTP URL 更改为 HTTPS，无需重定向。

集群管理员可将 HSTS 配置为执行以下操作：

- 根据每个路由启用 HSTS
- 根据每个路由禁用 HSTS
- 对一组域强制每个域的 HSTS，或者结合使用命名空间标签与域

重要

HSTS 仅适用于安全路由，可以是 edge-terminated 或 re-encrypt。其配置在 HTTP 或传递路由上无效。

6.1.3.1. 根据每个路由启用 HTTP 严格传输安全性

HTTP 严格传输安全 (HSTS) 实施在 HAProxy 模板中，并应用到具有 **haproxy.router.openshift.io/hsts_header** 注解的边缘和重新加密路由。

先决条件

- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 **oc** CLI。

流程

- 要在路由上启用 HSTS，请将 **haproxy.router.openshift.io/hsts_header** 值添加到 edge-terminated 或 re-encrypt 路由中。您可以运行以下命令来使用 **oc annotate** 工具来实现此目的：

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

1 在本例中，最长期限设置为 **31536000** ms，大约有 8 小时和半小时。

注意

在这个示例中，等号(=)是引号里。这是正确执行注解命令所必需的。

配置了注解的路由示例

```
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. 必需。**Max-age** 测量 HSTS 策略生效的时间长度，以秒为单位。如果设置为 **0**，它将对策略进行求反。
2. 可选。包含时，**includeSubDomains** 告知客户端主机的所有子域都必须与主机具有相同的 HSTS 策略。
3. 可选。当 **max-age** 大于 0 时，您可以在 **haproxy.router.openshift.io/hsts_header** 中添加 **preload**，以允许外部服务将这个站点包括在 HSTS 预加载列表中。例如，Google 等站点可以构造设有 **preload** 的站点的列表。浏览器可以使用这些列表来确定哪些站点可以通过 HTTPS 通信，即使它们与站点交互之前也是如此。如果没有设置 **preload**，浏览器必须已经通过 HTTPS 与站点交互（至少一次）才能获取标头。

6.1.3.2. 根据每个路由禁用 HTTP 严格传输安全性

要禁用 HTTP 严格传输安全性 (HSTS)，您可以将路由注解中的 **max-age** 值设置为 **0**。

先决条件

- 您可以使用具有项目的管理员特权的用户登陆到集群。
- 已安装 **oc** CLI。

流程

- 要禁用 HSTS，请输入以下命令将路由注解中的 **max-age** 值设置为 **0** ：

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

提示

您还可以应用以下 YAML 来创建配置映射：

根据每个路由禁用 HSTS 的示例

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

- 要为命名空间中的所有路由禁用 HSTS，请输入以下命令：

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

验证

1. 要查询所有路由的注解，请输入以下命令：

```
$ 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}}'
```

输出示例

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

6.1.4. 使用 Cookie 来保持路由有状态性

OpenShift Dedicated 提供粘性会话，通过确保所有流量都到达同一端点来实现有状态应用程序流量。但是，如果端点 pod 以重启、扩展或更改配置的方式被终止，这种有状态性可能会消失。

OpenShift Dedicated 可以使用 Cookie 来配置会话持久性。Ingress Controller 选择一个端点来处理任何用户请求，并为会话创建一个 Cookie。Cookie 在响应请求时返回，用户则通过会话中的下一请求发回 Cookie。Cookie 告知 Ingress Controller 哪个端点正在处理会话，确保客户端请求使用这个 Cookie 使请求路由到同一个 pod。

注意

无法在 passthrough 路由上设置 Cookie，因为无法看到 HTTP 流量。相反，根据源 IP 地址计算数字，该地址决定了后端。

如果后端更改，可以将流量定向到错误的服务器，使其更不计。如果您使用负载均衡器来隐藏源 IP，则会为所有连接和流量都发送到同一 pod 设置相同的数字。

6.1.4.1. 使用 Cookie 标注路由

您可以设置 Cookie 名称来覆盖为路由自动生成的默认名称。这样，接收路由流量的应用程序就能知道 Cookie 名称。通过删除 Cookie，它可以强制下一请求重新选择端点。因此，如果服务器过载，它会尝试从客户端中删除请求并重新分发它们。

流程

1. 使用指定的 Cookie 名称标注路由：

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

其中：

<route_name>
指定路由的名称。

<cookie_name>
指定 Cookie 的名称。

例如，使用 cookie 名称 **my_cookie** 标注路由 **my_route**：

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

2. 在变量中捕获路由主机名：

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

其中：

<route_name>
指定路由的名称。

3. 保存 cookie，然后访问路由：

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

使用上一个命令在连接到路由时保存的 cookie：

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

6.1.5. 基于路径的路由

基于路径的路由指定了一个路径组件，可以与 URL 进行比较，该 URL 需要基于 HTTP 的路由流量。因此，可以使用同一主机名提供多个路由，每个主机名都有不同的路径。路由器应该匹配基于最具体路径的路由。不过，这还取决于路由器的实现。

下表显示了路由及其可访问性示例：

表 6.1. 路由可用性

Route（路由）	当比较到	可访问
www.example.com/test	*www.example.com/test*	是
	www.example.com	否
www.example.com/test 和 *www.example.com*	*www.example.com/test*	是
	www.example.com	是
www.example.com	*www.example.com/text*	yes（由主机匹配，而不是路由）
	www.example.com	是

带有路径的未安全路由

```
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
```

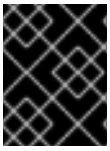
1 该路径是基于路径的路由的唯一添加属性。

注意

使用 passthrough TLS 时，基于路径的路由不可用，因为路由器不会在这种情况下终止 TLS，且无法读取请求的内容。

6.1.6. 特定于路由的注解

Ingress Controller 可以为它公开的所有路由设置默认选项。单个路由可以通过在其注解中提供特定配置来覆盖这些默认设置。红帽不支持在 Operator 管理的路由中添加路由注解。



重要

要创建带有多个源 IP 或子网的白名单，请使用以空格分隔的列表。任何其他限定类型会导致忽略列表，而不发出警告或错误消息。

表 6.2. 路由注解

变量	描述	默认的环境变量
haproxy.router.openshift.io/balance	设置负载平衡算法。可用选项是 **random**、**source**、**roundrobin** 和 **leastconn**。默认值为 **random**。	passthrough 路由 使用 **ROUTER_TCP_BALANCE_SCHEME** 。否则，使用 **ROUTER_LOAD_BALANCE_algorithm**。
haproxy.router.openshift.io/disable_cookies	禁用使用 cookie 来跟踪相关连接。如果设置为 **'true'** 或 **'TRUE'**，则使用均衡算法选择每个传入 HTTP 请求的后端服务连接。	
router.openshift.io/cookie_name	指定一个可选的、用于此路由的 cookie。名称只能包含大写字母和小写字母、数字、"_" 和 "-"。默认为路由的内部密钥进行哈希处理。	
haproxy.router.openshift.io/pod-concurrent-connections	设置路由器支持的 pod 允许的最大连接数。 注：如果有多个 pod，每个 pod 都有这些数量的连接。如果有多个路由器，它们之间没有协调关系，每个路由器都可能会多次连接。如果没有设置，或者将其设定为 0，则没有限制。	

变量	描述	默认的环境变量
haproxy.router.openshift.io/rate-limit-connections	设置 **'true'** 或 **'TRUE'** 可启用速率限制功能，该功能通过每个路由上的特定后端的贴子实施。 注：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	限制通过同一源 IP 地址进行的并发 TCP 连接数。它接受一个数字值。 注：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	限制具有相同源 IP 地址的客户端可以发出 HTTP 请求的速率。它接受一个数字值。 注：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	限制具有相同源 IP 地址的客户端可以进行 TCP 连接的速率。它接受一个数字值。 注：使用此注解可提供基本保护，防止分布式拒绝服务 (DDoS) 攻击。	
haproxy.router.openshift.io/timeout	为路由设定服务器端超时。(TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
haproxy.router.openshift.io/timeout-tunnel	这个超时适用于隧道连接，如明文、边缘、重新加密或透传路由。使用明文、边缘或重新加密路由类型，此注解作为带有现有超时值的超时隧道应用。对于 passthrough 路由类型，注解优先于设置任何现有的超时值。	**ROUTER_DEFAULT_TUNNEL_TIMEOUT**
ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after	您可以设置 IngressController 或 ingress 配置。此注解重新部署路由器，并将 HA 代理配置为在全局后发出 haproxy **hard-stop- after** 全局 选项，用于定义执行干净的软停止的最长时间。	**ROUTER_HARD_STOP_AFTER**
router.openshift.io/haproxy.health.check.interval	为后端健康检查设定间隔。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**

变量	描述	默认的环境变量
haproxy.router.openshift.io/ip_whitelist	为路由设置白名单。白名单是以空格分开的 IP 地址和 CIDR 范围列表，用来代表批准的源地址。来自白名单以外的 IP 地址的请求会被丢弃。 白名单中允许的最大 IP 地址和 CIDR 范围数为 61。	
haproxy.router.openshift.io/hsts_header	为 edge terminated 或 re-encrypt 路由设置 Strict-Transport-Security 标头。	
haproxy.router.openshift.io/log-send-hostname	在 Syslog 标头中设置 **hostname** 字段。使用系统的主机名。如果路由器启用了任何 Ingress API 日志记录方法（如 sidecar 或 Syslog 工具），则默认启用 **log-send-hostname**。	
haproxy.router.openshift.io/rewrite-target	在后端中设置请求的重写路径。	
router.openshift.io/cookie-same-site	设置一个值来限制 cookies。数值是： **Lax**：cookies 在访问的站点和第三方站点间进行传输。 **Strict**：cookies 仅限于访问的站点。 **None**：cookies 仅限于指定的站点。 这个值仅适用于重新加密和边缘路由。如需更多信息，请参阅 SameSite cookies 文档。	

变量	描述	默认的环境变量
haproxy.router.openshift.io/set-forwarded-headers	设置用于处理每个路由的 **Forwarded** 和 **X-Forwarded-For** HTTP 标头的策略。数值是： **Append**附加标头，保留任何现有的标头。这是默认值。 **replace**：设置标头，删除任何现有的标头。 **Never**：不设置标头，而是保留任何现有的标头。 **if-none**：如果没有设置标头，则设置它。	**ROUTER_SET_FORWARDED_HEADERS**

注意

环境变量不能编辑。

路由器超时变量

TimeUnits 由一个数字及一个时间单位表示：**us** *(microseconds), **ms**（毫秒，默认）、**s**（秒）、**m**（分钟）、**h** *(小时)、**d** （天）。

正则表达式是： [1-9][0-9]*(**us**\|**ms**\|**s**\|**m**\|**h**\|**d**)。

变量	默认	Description
ROUTER_BACKEND_CHECK_INTERVAL	**5000ms**	后端上后续存活度检查之间的时长。
ROUTER_CLIENT_FIN_TIMEOUT	**1s**	控制连接到路由的客户端的 TCP FIN 超时周期。如果发送到关闭连接的 FIN 在给定时间内没有回答，HAProxy 会关闭连接。如果设置为较低值，并且在路由器上使用较少的资源，则这不会产生任何损害。
ROUTER_DEFAULT_CLIENT_TIMEOUT	**30s**	客户端必须确认或发送数据的时长。
ROUTER_DEFAULT_CONNECT_TIMEOUT	**5s**	最长连接时间。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	**1s**	控制路由器到支持路由的 pod 的 TCP FIN 超时。
ROUTER_DEFAULT_SERVER_TIMEOUT	**30s**	服务器必须确认或发送数据的时长。

变量	默认	Description
ROUTER_DEFAULT_TUNNEL_TIMEOUT	**1h**	TCP 或 WebSocket 连接保持打开的时长。每当 HAProxy 重新加载时，这个超时期限都会重置。
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	**300s**	设置等待出现新 HTTP 请求的最长时间。如果设置得太低，可能会导致浏览器和应用程序无法期望较小的 **keepalive** 值。 某些有效的超时值可以是某些变量的总和，而不是特定的预期超时。例如：**ROUTER_SLOWLORIS_HTTP_KEEPALIVE** 调整 **timeout http-keep-alive**。默认情况下，它设置为 **300s**，但 HAProxy 也会在 **tcp-request inspect-delay** 上等待，它被设置为 **5s**。在这种情况下，整个超时时间将是 **300s** 加 **5s**。
ROUTER_SLOWLORIS_TIMEOUT	**10s**	HTTP 请求传输可以花费的时间长度。
RELOAD_INTERVAL	**5s**	允许路由器至少执行重新加载和接受新更改的频率。
ROUTER_METRICS_HAPROXY_TIMEOUT	**5s**	收集 HAProxy 指标的超时时间。

设置自定义超时的路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
...
```

(1) 使用 HAProxy 支持的时间单位（**us**, **ms**, **s**, **m**, **h**, **d**）指定新的超时时间。如果没有提供时间单位，**ms** 会被默认使用。

注意

如果为 passthrough 路由设置的服务器端的超时值太低，则会导致 WebSocket 连接在那个路由上经常出现超时的情况。

只允许一个特定 IP 地址的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

允许多个 IP 地址的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

允许 IP 地址 CIDR 网络的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

允许 IP 地址和 IP 地址 CIDR 网络的路由

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

指定重写对象的路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / (1)
...
```

(1) 将 / 设为后端请求的重写路径。

在路由上设置 **haproxy.router.openshift.io/rewrite-target** 注解，指定 Ingress Controller 在将请求转发到后端应用程序之前，应该使用此路由在 HTTP 请求中重写路径。与 **spec.path** 中指定的路径匹配的请求路径部分将替换为注解中指定的重写对象。

下表提供了在 **spec.path**、请求路径和重写对象的各种组合中重写行为的路径示例。

表 6.3. rewrite-target 示例：

Route.spec.path	请求路径	重写目标	转发请求路径
/foo	/foo	/	/
/foo	/foo/	/	/
/foo	/foo/bar	/	/bar
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/

Route.spec.path	请求路径	重写目标	转发请求路径
/foo	/foo/bar	/baz	/baz/bar
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	不适用（请求路径不匹配路由路径）
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

6.1.7. 通过 Ingress 对象使用默认证书创建路由

如果您在没有指定 TLS 配置的情况下创建 Ingress 对象，OpenShift Dedicated 会生成不安全的路由。要创建使用默认入口证书生成安全边缘终止路由的 Ingress 对象，您可以指定一个空的 TLS 配置，如下所示。

先决条件

- 您有一个要公开的服务。
- 您可以访问 OpenShift CLI(**oc**)。

流程

1. 为 Ingress 对象创建 YAML 文件。在本例中，该文件名为 **example-ingress.yaml** ：

   Ingress 对象的 YAML 定义

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

   1 使用此准确语法指定 TLS，而不指定自定义证书。

2. 运行以下命令来创建 Ingress 对象：

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

验证

- 运行以下命令，验证 OpenShift Dedicated 是否为 Ingress 对象创建了预期的路由：

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

输出示例

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

1. 路由的名称包括 Ingress 对象的名称，后跟一个随机后缀。
2. 要使用默认证书，路由不应指定 **spec.certificate**。
3. 路由应该指定 **边缘终止** 策略。

6.1.8. 在 Ingress 注解中使用目标 CA 证书创建路由

在 Ingress 对象上可以使用 **route.openshift.io/destination-ca-certificate-secret** 注解来定义带有自定义目标 CA 证书的路由。

先决条件

- 您可以在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须在 PEM 编码文件中有单独的目标 CA 证书。
- 您必须具有要公开的服务。

流程

1. 将 **route.openshift.io/destination-ca-certificate-secret** 添加到 Ingress 注解中：

```
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 该注解引用 kubernetes secret。

2. 此注解中引用的机密将插入到生成的路由中。

输出示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

6.1.9. 为双栈网络配置 OpenShift Dedicated Ingress Controller

如果您的 OpenShift Dedicated 集群是为 IPv4 和 IPv6 双栈网络配置的，则 OpenShift Dedicated 路由可从外部访问集群。

Ingress Controller 会自动提供具有 IPv4 和 IPv6 端点的服务，但您可以为单堆栈或双栈服务配置 Ingress Controller。

先决条件

- 您在裸机上部署了 OpenShift Dedicated 集群。
- 已安装 OpenShift CLI（**oc**）。

流程

1. 要使 Ingress Controller 为工作负载提供通过 IPv4/IPv6 的流量，您可以通过设置 ipFamilies 和 ipFamilyPolicy 字段来创建服务 YAML 文件，或通过设置 **ipFamilies** 和 **ipFamilyPolicy** 字段来修改现有服务 YAML 文件。例如：

服务 YAML 文件示例

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: yyyy-mm-ddT00:00:00Z
  labels:
    name: <service_name>
    manager: kubectl-create
    operation: Update
```

```
    time: yyyy-mm-ddT00:00:00Z
  name: <service_name>
  namespace: <namespace_name>
  resourceVersion: "<resource_version_number>"
  selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
  uid: <uid_number>
spec:
  clusterIP: 172.30.0.0/16
  clusterIPs: 1
  - 172.30.0.0/16
  - <second_IP_address>
  ipFamilies: 2
  - IPv4
  - IPv6
  ipFamilyPolicy: RequireDualStack 3
  ports:
  - port: 8080
    protocol: TCP
    targetport: 8080
  selector:
    name: <namespace_name>
  sessionAffinity: None
  type: ClusterIP
status:
  loadbalancer: {}
```

1. 在双栈实例中，提供了两个不同的 **clusterIP**。
2. 对于单堆栈实例，输入 **IPv4** 或 **IPv6**。对于双栈实例，请输入 **IPv4** 和 **IPv6**。
3. 对于单堆栈实例，请输入 **SingleStack**。对于双栈实例，请输入 **RequireDualStack**。

这些资源生成对应的**端点**。Ingress Controller 现在监视 **endpointslices**。

2. 要查看**端点**，请输入以下命令：

```
$ oc get endpoints
```

3. 要查看**endpointslices**，输入以下命令：

```
$ oc get endpointslices
```

其他资源

- 使用 appsDomain 选项指定备选集群域

6.2. 安全路由

安全路由提供以下几种 TLS 终止功能来为客户端提供证书。以下小节介绍了如何使用自定义证书创建重新加密、边缘和透传路由。

重要

如果您在 Microsoft Azure 中创建通过公共端点的路由，则资源名称会受到限制。您不能创建使用某些词语的资源。如需 Azure 限制词语的列表，请参阅 Azure 文档中的解决预留资源名称错误。

6.2.1. 使用自定义证书创建重新加密路由

您可以通过 **oc create route** 命令，使用重新加密 TLS 终止和自定义证书配置安全路由。

先决条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须在 PEM 编码文件中有单独的目标 CA 证书。
- 您必须具有要公开的服务。

注意

不支持密码保护的密钥文件。要从密钥文件中删除密码，使用以下命令：

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

流程

此流程使用自定义证书和重新加密 TLS 终止创建 **Route** 资源。以下步骤假定证书/密钥对位于当前工作目录下的 **tls.crt** 和 **tls.key** 文件中。您还必须指定一个目标 CA 证书，使 Ingress Controller 信任服务的证书。您也可以根据需要指定 CA 证书来补全证书链。替换 **tls.crt**、**tls.key**、**cacert.crt** 和（可选）**ca.crt** 的实际路径名称。替换您要为 **frontend** 公开的 **Service** 资源的名称。使用适当的主机名替换 **www.example.com**。

- 使用重新加密 TLS 终止和自定义证书，创建安全 **Route** 资源：

```
$ 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
```

如果您检查生成的 **Route** 资源，它应该类似于如下：

安全路由 YAML 定义

```
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-----
```

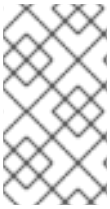
如需了解更多选项，请参阅 **oc create route reencrypt --help**。

6.2.2. 使用自定义证书创建边缘路由

您可以通过 **oc create route** 命令，使用边缘 TLS 终止和自定义证书配置安全路由。使用边缘路由时，Ingress Controller 在将流量转发到目标 pod 之前终止 TLS 加密。该路由指定了 Ingress Controller 用于路由的 TLS 证书和密钥。

先决条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中的证书对路由主机有效。
- 您可以在 PEM 编码文件中有一个单独的 CA 证书来补全证书链。
- 您必须具有要公开的服务。



注意

不支持密码保护的密钥文件。要从密钥文件中删除密码，使用以下命令：

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

流程

此流程使用自定义证书和边缘 TLS 终止创建 **Route** 资源。以下步骤假定证书/密钥对位于当前工作目录下的 **tls.crt** 和 **tls.key** 文件中。您也可以根据需要指定 CA 证书来补全证书链。替换 **tls.crt**、**tls.key** 和（可选）**ca.crt** 的实际路径名称。替换您要为 **frontend** 公开的服务名称。使用适当的主机名替换 **www.example.com**。

- 使用边缘 TLS 终止和自定义证书，创建安全 **Route** 资源。

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

如果您检查生成的 **Route** 资源，它应该类似于如下：

安全路由 YAML 定义

```
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-----
```

如需了解更多选项，请参阅 **oc create route edge --help**。

6.2.3. 创建 passthrough 路由

您可以使用 **oc create route** 命令使用 passthrough 终止配置安全路由。如果 passthrough 终止，加密的流量会直接发送到目的地，而路由器不会提供 TLS 终止。因此，路由不需要密钥或证书。

先决条件

- 您必须具有要公开的服务。

流程

- 创建 **Route** 资源：

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

如果您检查生成的 **Route** 资源，它应该类似于如下：

使用 Passthrough 终止的安全路由

```
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 对象的名称，长度限于 63 个字符。

2 **termination** 字段设置为 **passthrough**。这是唯一需要 **tls** 的字段。

3 可选的 **insecureEdgeTerminationPolicy**。禁用后唯一有效的值是 **None**、**Redirect** 或为空。

目标 pod 负责为端点上的流量提供证书。目前，这是唯一支持需要客户端证书的方法，也称双向验证。