OpenShift Dedicated 4

网络

配置 OpenShift Dedicated 网络

Last Updated: 2025-03-23

OpenShift Dedicated 4 网络

配置 OpenShift Dedicated 网络

法律通告

摘要

本文档提供有关 OpenShift Dedicated 集群联网的信息。

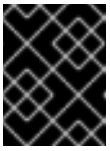
目录

第 1 章 关于网络

Red Hat OpenShift 网络是一个功能、插件和高级网络功能生态系统，它增强了 Kubernetes 网络与高级网络相关的功能，集群需要管理一个或多个混合集群的网络流量。这种网络功能生态系统集成了入口、出口、负载均衡、高性能吞吐量、安全性和集群内部流量管理。Red Hat OpenShift 网络生态系统还提供基于角色的可观察工具来降低其自然复杂性。

以下是集群中一些最常用的 Red Hat OpenShift Networking 功能：

- 用于网络插件管理的 Cluster Network Operator
- 由以下 Container Network Interface (CNI) 插件之一提供的主要集群网络：
  - OVN-Kubernetes 网络插件，这是默认的 CNI 插件。
  - OpenShift SDN 网络插件，该插件在 OpenShift 4.16 中已弃用，并在 OpenShift 4.17 中删除。



重要

您无法将使用 SDN 网络插件的 OpenShift 4.16 集群迁移到 OpenShift 4.17，因为当前不存在迁移路径。

其他资源

- OpenShift SDN CNI 在 OCP 4.17 中删除

第 2 章 网络 OPERATOR

2.1. OPENSHIFT DEDICATED 中的 DNS OPERATOR

在 OpenShift Dedicated 中，DNS Operator 部署和管理 CoreDNS 实例，为集群中的 pod 提供名称解析服务，启用基于 DNS 的 Kubernetes 服务发现，并解析内部 **cluster.local** 名称。

2.1.1. 检查 DNS Operator 的状态

DNS Operator 从 **operator.openshift.io** API 组实现 **dns** API。Operator 使用守护进程集部署 CoreDNS，为守护进程集创建一个服务，并将 kubelet 配置为指示 pod 使用 CoreDNS 服务 IP 地址进行名称解析。

流程

在安装过程中使用 **Deployment** 对象部署 DNS Operator。

1. 使用 **oc get** 命令查看部署状态：

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

输出示例

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

2. 使用 **oc get** 命令来查看 DNS Operator 的状态：

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

输出示例

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

AVAILABLE、**PROGRESSING** 和 **DEGRADED** 提供了有关 Operator 状态的信息。当 CoreDNS 守护进程中至少有一个 pod 被设置为 **Available** 状态时，**AVAILABLE** 为 **True**，DNS 服务有一个集群 IP 地址。

2.1.2. 查看默认 DNS

每个新的 OpenShift Dedicated 安装都有一个名为 **default** 的 **dns.operator**。

流程

1. 使用 **oc describe** 命令来查看默认 **dns**：

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

输出示例

```
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 字段是用来构造完全限定的 pod 和服务域名的基本 DNS 域。
2. Cluster IP 是 Pod 为名称解析查询的地址。IP 由服务 CIDR 范围中的第 10 个地址定义。

2.1.3. 使用 DNS 转发

您可以使用以下方法使用 DNS 转发来覆盖 **/etc/resolv.conf** 文件中的默认转发配置：

- 为每个区指定名称服务器 (**spec.servers**)。如果转发区是 OpenShift Dedicated 管理的入口域，则上游名称服务器必须为域授权。

重要

您必须至少指定一个区。否则，集群可能会丢失功能。

- 提供上游 DNS 服务器列表 (**spec.upstreamResolvers**)。
- 更改默认转发策略。

注意

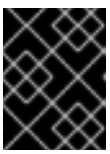
默认域的 DNS 转发配置可以同时在 **/etc/resolv.conf** 文件和上游 DNS 服务器中指定默认服务器。

流程

1. 修改名为 **default** 的 DNS Operator 对象：

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

发出上一命令后，Operator 会根据 **spec.servers** 创建并更新名为 **dns-default** 的配置映射，并使用额外的服务器配置块。



重要

当为 **zones** 参数指定值时，请确保只转发到特定区域，如您的内网。您必须至少指定一个区。否则，集群可能会丢失功能。

如果任何服务器都没有与查询匹配的区域，则名称解析会返回上游 DNS 服务器。

配置 DNS 转发

```
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. 必须符合 **rfc6335** 服务名称语法。
2. 必须符合 **rfc1123** 服务名称语法中的子域的定义。集群域 **cluster.local** 是对 **zones** 字段的无效子域。
3. 定义用于选择 **forwardPlugin** 中列出的上游解析器的策略。默认值为 **Random**。您还可以使用 **RoundRobin**, 和 **Sequential** 值。
4. 每个 **forwardPlugin** 最多允许 15 个 **upstreams**。
5. 您可以使用 **upstreamResolvers** 覆盖默认转发策略，并将 DNS 解析转发到默认域的指定 DNS 解析器（上游解析器）。如果没有提供任何上游解析器，DNS 名称查询将进入 **/etc/resolv.conf** 中声明的服务器。
6. 决定选择上游中列出的 **upstreams** 服务器进行查询的顺序。您可以指定这些值之一：**Random**、**RoundRobin** 或 **Sequential**。默认值为 **Sequential**。
7. 如果被省略，平台会选择一个默认值，通常是原始客户端请求的协议。设置为 **TCP**，以指定平台应该对所有上游 DNS 请求使用 TCP，即使客户端请求使用了 UDP。

8 用于配置传输类型、服务器名称和可选自定义 CA 或 CA 捆绑包，以便在将 DNS 请求转发到上游解析器时使用。

9 您可以指定两个类型的 **upstreams**：**SystemResolvConf** 或 **Network**。**SystemResolvConf** 将上游配置为使用 **/etc/resolv.conf** 和 **Network** 定义一个 **Networkresolver**。您可以指定其中一个或两者都指定。

10 如果指定类型是 **Network**，则必须提供 IP 地址。**address** 字段必须是有效的 IPv4 或 IPv6 地址。

11 如果指定类型是 **Network**，您可以选择性地提供端口。**port** 字段必须是 **1** 到 **65535** 之间的值。如果您没有为上游指定端口，则默认端口为 853。

其他资源

- 有关 DNS 转发的详情，请查看 CoreDNS 转发文档。

2.1.4. 检查 DNS Operator 状态

您可以使用 **oc describe** 命令来检查状态并查看 DNS Operator 的详情。

流程

- 查看 DNS Operator 的状态：

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

虽然在特定版本中的信息和拼写可能有所不同，但预期的状态输出如下：

```
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
```

2.1.5. 查看 DNS Operator 日志

您可以使用 **oc logs** 命令来查看 DNS Operator 日志。

流程

- 查看 DNS Operator 的日志：

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

2.1.6. 设置 CoreDNS 日志级别

CoreDNS 和 CoreDNS Operator 的日志级别使用不同的方法设置。您可以配置 CoreDNS 日志级别来确定日志记录错误信息中的详情量。CoreDNS 日志级别的有效值为 **Normal**、**Debug** 和 **Trace**。默认 **logLevel** 为 **Normal**。

注意

CoreDNS 错误日志级别总是被启用。以下日志级别设置报告不同的错误响应：

- **logLevel**: **Normal** 启用 "errors" 类: **log . { class error }**.
- **loglevel** ：**Debug** 启用 "denial" 类: **log . { class denial error }**。
- **logLevel**: **Trace** 启用 "all" 类: **log . { class all }**.

流程

- 要将 **logLevel** 设置为 **Debug**，输入以下命令：

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

- 要将 **logLevel** 设置为 **Trace**，输入以下命令：

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

验证

- 要确保设置了所需的日志级别，请检查配置映射：

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

例如，在将 **logLevel** 设置为 **Trace** 后，您应该在每个 server 块中看到这个小节：

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

2.1.7. 设置 CoreDNS Operator 的日志级别

CoreDNS 和 CoreDNS Operator 的日志级别使用不同的方法设置。集群管理员可以配置 Operator 日志级别来更快地跟踪 OpenShift DNS 问题。**operatorLogLevel** 的有效值为 **Normal**、**Debug** 和 **Trace**。**Trace** 具有更详细的信息。默认 **operatorlogLevel** 为 **Normal**。Operator 问题有七个日志记录级

别： Trace, Debug, Info, Warning, Error, Fatal, 和 Panic。设置了日志级别后，具有该严重性级别或以上级别的所有内容都会记录为日志条目。

- **operatorLogLevel: "Normal"** 设置 **logrus.SetLogLevel("Info")**。
- **operatorLogLevel: "Debug"** 设置 **logrus.SetLogLevel("Debug")**。
- **operatorLogLevel: "Trace"** 设置 **logrus.SetLogLevel("Trace")**。

流程

- 要将 **operatorLogLevel** 设置为 **Debug**，请输入以下命令：

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

- 要将 **operatorLogLevel** 设置为 **Trace**，请输入以下命令：

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

验证

1. 要查看生成的更改，请输入以下命令：

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

   您应该会看到两个日志级别条目。**operatorLogLevel** 适用于 OpenShift DNS Operator 问题，**logLevel** 适用于 CoreDNS pod 的 daemonset：

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

2. 要查看 daemonset 的日志，请输入以下命令：

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

2.1.8. 调整 CoreDNS 缓存

对于 CoreDNS，您可以配置成功或不成功缓存的最长持续时间，也称为正缓存或负缓存。调整 DNS 查询响应的缓存持续时间可减少任何上游 DNS 解析器的负载。

警告

将 TTL 字段设为低值可能会导致集群、任何上游解析器或两者中负载的增加。

流程

1. 运行以下命令来编辑名为 **default** 的 DNS Operator 对象：

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

2. 修改生存时间 (TTL) 缓存值：

配置 DNS 缓存

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

1 CoreDNS 会将字符串值 **1h** 转换为其相应的秒数。如果省略此字段，则假定该值为 **0s**，集群将使用内部默认值 **900s** 作为回退。

2 字符串值可以是单元的组合（如 **0.5h10m**），并被 CoreDNS 转换为相应秒数。如果省略了此字段，则假定该值为 **0s**，集群将使用内部默认值 **30s** 作为回退。

验证

1. 要查看更改，请运行以下命令再次查看配置映射：

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

2. 验证您是否看到类似以下示例的条目：

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

其他资源

有关缓存的更多信息，请参阅 CoreDNS 缓存。

2.1.9. 高级任务

2.1.9.1. 更改 DNS Operator managementState

DNS Operator 管理 CoreDNS 组件，为集群中的 pod 和服务提供名称解析服务。默认情况下，DNS Operator 的 **managementState** 设置为 **Managed**，这意味着 DNS Operator 会主动管理其资源。您可以将其更改为 **Unmanaged**，这意味着 DNS Operator 不管理其资源。

以下是更改 DNS Operator **managementState** 的用例：

- 您是一个开发者，希望测试配置更改来查看它是否解决了 CoreDNS 中的问题。您可以通过将 **managementState** 设置为 **Unmanaged** 来停止 DNS Operator 覆盖配置更改。

- 您是一个集群管理员，报告了 CoreDNS 的问题，但在解决这个问题前需要应用一个临时解决方案。您可以将 DNS Operator 的 **managementState** 字段设置为 **Unmanaged** 以应用临时解决方案。

流程

1. 在 DNS Operator 中将 **managementState** 更改为 **Unmanaged** ：

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

2. 使用 **jsonpath** 命令行 JSON 解析器查看 DNS Operator 的 **managementState** ：

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

输出示例

```
"Unmanaged"
```

注意

当 **managementState** 设置为 **Unmanaged** 时，您无法升级。

2.1.9.2. 控制 DNS pod 放置

DNS Operator 有两个守护进程集：一个用于 CoreDNS，一个用于管理名为 **node - resolver** 的 **/etc/hosts** 文件。

您可能会发现，需要控制哪些节点已分配并运行了 CoreDNS pod，尽管这不是一个常见操作。例如，如果集群管理员配置了可以禁止节点对节点间通信的安全策略，这需要限制 CoreDNS 运行 daemonset 的节点集合。如果 DNS pod 在集群中的某些节点上运行，且没有运行 DNS pod 的节点与运行 DNS pod 的节点具有网络连接，则 DNS 服务将适用于所有 pod。

node-resolver 守护进程集必须在每个节点主机上运行，因为它为集群镜像 registry 添加一个条目来支持拉取镜像。**node-resolver** pod 只有一个作业：查找 **image-registry.openshift-image-registry.svc** 服务的集群 IP 地址，并将其添加到节点主机上的 **/etc/hosts** 中，以便容器运行时可以解析服务名称。

作为集群管理员，您可以使用自定义节点选择器将 CoreDNS 的守护进程集配置为在某些节点上运行或不运行。

先决条件

- 已安装 **oc** CLI。
- 以具有 **cluster-admin** 权限的用户身份登录集群。
- 您的 DNS Operator **managementState** 设置为 **Managed**。

流程

- 要允许 CoreDNS 的守护进程集在特定节点上运行，请配置污点和容限：
  1. 修改名为 **default** 的 DNS Operator 对象：

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

2. 为污点指定污点键和一个容忍度：

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

1 如果污点是 **dns-only**，它可以无限期地被容许。您可以省略 **tolerationSeconds**。

2.1.9.3. 使用 TLS 配置 DNS 转发

在高度监管的环境中工作时，您可能需要在将请求转发到上游解析器时保护 DNS 流量，以便您可以确保额外的 DNS 流量和数据隐私。

请注意，CoreDNS 缓存在 10 秒内转发连接。如果没有请求，CoreDNS 将为该 10 秒打开 TCP 连接。对于大型集群，请确保您的 DNS 服务器知道可能有多个新的连接来保存打开，因为您可以在每个节点上启动连接。相应地设置 DNS 层次结构以避免性能问题。

重要

当为 **zones** 参数指定值时，请确保只转发到特定区域，如您的内网。您必须至少指定一个区。否则，集群可能会丢失功能。

流程

1. 修改名为 **default** 的 DNS Operator 对象：

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

集群管理员可以配置传输层安全(TLS)来转发 DNS 查询。

使用 TLS 配置 DNS 转发

```
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. 必须符合 **rfc6335** 服务名称语法。
2. 必须符合 **rfc1123** 服务名称语法中的子域的定义。集群域 **cluster.local** 是对 **zones** 字段的无效子域。集群域 **cluster.local** 不是 **zones** 中的一个有效的 **subdomain**。
3. 在为转发 DNS 查询配置 TLS 时，将 **transport** 字段设置为具有值 **TLS**。
4. 当为转发 DNS 查询配置 TLS 时，这是用作服务器名称的一部分(SNI)的强制服务器名称来验证上游 TLS 服务器证书。
5. 定义用于选择上游解析器的策略。默认值为 **Random**。您还可以使用 **RoundRobin**, 和 **Sequential** 值。
6. 必需。使用它提供上游解析器。每个 **forwardPlugin** 条目最多允许 15 个 **upstreams** 条目。
7. 可选。您可以使用它来覆盖默认策略，并将 DNS 解析转发到默认域的指定 DNS 解析器（上游解析器）。如果没有提供任何上游解析器，DNS 名称查询将进入 **/etc/resolv.conf** 中的服务器。
8. 在使用 TLS 时，只允许**网络**类型，且您必须提供 IP 地址。**网络**类型表示，该上游解析器应该独立于 **/etc/resolv.conf** 中列出的上游解析器单独处理转发请求。
9. **address** 字段必须是有效的 IPv4 或 IPv6 地址。
10. 您可以选择提供端口。**port** 必须是 **1** 到 **65535** 之间的值。如果您没有为上游指定端口，则默认端口为 853。

注意

如果 **servers** 未定义或无效，则配置映射只包括默认服务器。

验证

1. 查看配置映射：

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

基于 TLS 转发示例的 DNS ConfigMap 示例

```
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 对 **forwardPlugin** 的更改会触发 CoreDNS 守护进程集的滚动更新。

其他资源

- 有关 DNS 转发的详情，请查看 CoreDNS 转发文档。

2.2. OPENSHIFT DEDICATED 中的 INGRESS OPERATOR

Ingress Operator 实现 **IngressController** API，是负责启用对 OpenShift Dedicated 集群服务的外部访问的组件。

2.2.1. OpenShift Dedicated Ingress Operator

在创建 OpenShift Dedicated 集群时，在集群中运行的 pod 和服务各自分配自己的 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 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 Reliability engineerss (SRE) 为 OpenShift Dedicated 集群管理

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。

2.2.2. Ingress 配置资产

安装程序在 **config.openshift.io** API 组中生成带有 **Ingress** 资源的资产，**cluster-ingress-02-config.yml**。

Ingress 资源的 YAML 定义

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

安装程序将这个资产保存在 **manifests/** 目录下的 **cluster-ingress-02-config.yml** 文件中。此 **Ingress** 资源定义 Ingress 的集群范围配置。此 Ingress 配置的用法如下所示：

- Ingress Operator 使用集群 Ingress 配置中的域，作为默认 Ingress Controller 的域。
- OpenShift API Server Operator 使用集群 Ingress 配置中的域。在为未指定显式主机的 **Route** 资源生成默认主机时，还会使用此域。

2.2.3. Ingress Controller 配置参数

IngressController 自定义资源(CR)包含可选配置参数，您可以配置它们来满足您的机构的特定需求。

参数	描述
domain	**domain** 是 Ingress Controller 服务的一个 DNS 名称，用于配置多个功能： ● 对于 **LoadBalancerService** 端点发布策略，**domain** 被用来配置 DNS 记录。请参阅 **endpointPublishingStrategy**。 ● 当使用生成的默认证书时，该证书对**域**及其**子域**有效。请参阅 **defaultCertificate**。 ● 该值会发布到独立的 Route 状态，以便用户了解目标外部 DNS 记录的位置。 **domain** 值在所有 Ingress 控制器中需要是唯一的，且不能更新。 如果为空，默认值为 **ingress.config.openshift.io/cluster .spec.domain**。
replicas	**replicas** 是 Ingress Controller 副本数量。如果没有设置，则默认值为**2**。

参数	描述
endpointPublishingStrategy	**endpointPublishingStrategy** 用于向其他网络发布 Ingress Controller 端点，以启用负载均衡器集成，并提供对其他系统的访问。 对于云环境，使用 **loadBalancer** 字段为 Ingress Controller 配置端点发布策略。 您可以配置以下 **endpointPublishingStrategy** 字段： ● **loadBalancer.scope** ● **loadBalancer.allowedSourceRanges** 如果没有设置，则默认值基于 **infrastructure.config.openshift.io/cluster .status.platform**： ● Amazon Web Services (AWS): **LoadBalancerService** (带有外部范围) ● Google Cloud Platform(GCP)： **LoadBalancerService** （具有外部范围） 对于大多数平台，可以更新 **endpointPublishingStrategy** 值。在 GCP 上，您可以配置以下 **endpointPublishingStrategy** 字段： ● **loadBalancer.scope** ● **loadbalancer.providerParameters.gcp.clientAccess** 如果需要在集群部署后更新 **endpointPublishingStrategy** 值，您可以配置以下 **endpointPublishingStrategy** 字段： ● **hostNetwork.protocol** ● **nodePort.protocol** ● **private.protocol**
defaultCertificate	**defaultCertificate** 的值是一个到包括由 Ingress controller 提供的默认证书的 secret 的指代。当 Routes 没有指定其自身证书时，使用 **defaultCertificate**。 secret 必须包含以下密钥和数据：* **tls.crt**：证书文件内容 * **tls.key**：密钥文件内容 如果没有设置，则自动生成和使用通配符证书。该证书对 Ingress Controller 的**域** 和**子域**有效，所生成的证书的 CA 会自动与集群的信任存储集成。 内部证书（无论是生成的证书还是用户指定的证书）自动与 OpenShift Dedicated 内置的 OAuth 服务器集成。
namespaceSelector	**namespaceSelector** 用来过滤由 Ingress 控制器提供服务的一组命名空间。这对实现分片（shard）非常有用。
routeSelector	**routeSelector** 用于由 Ingress Controller 提供服务的一组 Routes。这对实现分片（shard）非常有用。

参数	描述
nodePlacement	**NodePlacement** 启用对 Ingress Controller 调度的显式控制。 如果没有设置，则使用默认值。  **注意** **nodePlacement** 参数包括两个部分: **nodeSelector** 和 **tolerations**。例如： nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists
tlsSecurityProfile	**tlsSecurityProfile** 指定 Ingress Controller 的 TLS 连接的设置。 如果没有设置，则默认值基于 **apiservers.config.openshift.io/cluster** 资源。 当使用 **Old**、**Intermediate** 和 **Modern** 配置集类型时，有效的配置集可能会在不同发行版本间有所改变。例如：使用在版本 **X.Y.Z** 中部署的 **Intermediate** 配置集，升级到版本 **X.Y.Z+1** 可能会导致新的配置集配置应用到 Ingress Controller，从而导致一个 rollout 操作。 Ingress Controller 的最低 TLS 版本是 **1.1**，最高 TLS 版本为 **1.3**。  **注意** 加密器和配置的安全配置集的最小 TLS 版本反映在 **TLSProfile** 状态中。  **重要** Ingress Operator 将 **Old** 或 **Custom** 配置集的 TLS **1.0** 转换为 **1.1**。

参数	描述
clientTLS	**clientTLS** 验证客户端对集群和服务的访问；因此，启用了 mutual TLS 身份验证。如果没有设置，则不启用客户端 TLS。 **clientTLS** 具有所需的子字段 **spec.clientTLS.clientCertificatePolicy** 和 **spec.clientTLS.ClientCA**。 **ClientCertificatePolicy** 子字段接受以下两个值之一：**Required** 或 **Optional**。**ClientCA** 子字段指定 openshift-config 命名空间中的配置映射。配置映射应包含 CA 证书捆绑包。 **AllowedSubjectPatterns** 是一个可选值，用于指定正则表达式列表，该列表与有效客户端证书上的可分辨名称匹配以过滤请求。正则表达式必须使用 PCRE 语法。至少一种模式必须与客户端证书的可分辨名称匹配；否则，入口控制器拒绝证书，并拒绝连接。如果没有指定，ingress 控制器不会根据可分辨的名称拒绝证书。
routeAdmission	**routeAdmission** 定义了处理新路由声明的策略，如允许或拒绝命名空间间的声明。 **namespaceOwnership** 描述了如何处理跨命名空间的主机名声明。默认为 **Strict**。 - **Strict**：不允许路由在命名空间间声明相同的主机名。 - **InterNamespaceAllowed** ：允许路由在命名空间间声明相同主机名的不同路径。 **wildcardPolicy** 描述了 Ingress Controller 如何处理采用通配符策略的路由。 - **WildcardsAllowed**：表示 Ingress Controller 允许采用任何通配符策略的路由。 - **WildcardsDisallowed**：表示 Ingress Controller 只接受采用 **None** 通配符策略的路由。将 **wildcardPolicy** 从 **WildcardsAllowed** 更新为 **WildcardsDisallowed**，会导致采用 **Subdomain** 通配符策略的已接受路由停止工作。这些路由必须重新创建为采用 **None** 通配符策略，让 Ingress Controller 重新接受。**WildcardsDisallowed** 是默认设置。

参数	描述

IngressControllerLogging	**logging** 定义了有关在哪里记录什么内容的参数。如果此字段为空，则会启用运行日志，但禁用访问日志。
	• **access** 描述了客户端请求的日志记录方式。如果此字段为空，则禁用访问日志。
	  ◦ **destination** 描述日志消息的目的地。
	  ▪ **type** 是日志的目的地类型：
	  • **Container** 指定日志应该进入 sidecar 容器。Ingress Operator 在 Ingress Controller pod 上配置名为 **logs** 的容器，并配置 Ingress Controller 以将日志写入容器。管理员应该配置一个自定义日志记录解决方案，从该容器读取日志。使用容器日志意味着，如果日志速率超过容器运行时或自定义日志解决方案的容量，则可能会出现日志丢失的问题。
	  • **Syslog** 指定日志发送到 Syslog 端点。管理员必须指定可以接收 Syslog 消息的端点。管理员应该已经配置了一个自定义 Syslog 实例。
	  ▪ **container** 描述了 **Container** 日志记录目的地类型的参数。目前没有容器日志记录参数，因此此字段必须为空。
	  ▪ **syslog** 描述了 **Syslog** 日志记录目的地类型的参数：
	  • **address** 是接收日志消息的 syslog 端点的 IP 地址。
	  • **port** 是接收日志消息的 syslog 端点的 UDP 端口号。
	  • **MaxLength** 是 syslog 消息的最大长度。它必须介于 **480** 到 **4096** 字节之间。如果此字段为空，则最大长度设置为默认值 **1024** 字节。
	  • **facility** 指定日志消息的 syslog 工具。如果该字段为空，则工具为 **local1**。否则,它必须指定一个有效的 syslog 工具：**kern**、**user**、**mail**、**daemon**、**auth**、**syslog**, **lpr**, **news**, **uucp**, **cron**, **auth2**, **ftp**, **ntp**, **audit**, **alert**, **cron2**, **local0**, **local1**、**local2**、**local3**。**local4**、**local5**、**local6** 或 **local7**。
	  ◦ **httpLogFormat** 指定 HTTP 请求的日志消息格式。如果此字段为空，日志消息将使用实现中的默认 HTTP 日志格式。有关 HAProxy 的默认 HTTP 日志格式，请参阅 HAProxy 文档。

参数	描述
httpHeaders	**httpHeaders** 为 HTTP 标头定义策略。 通过为 **IngressControllerHTTPHeaders** 设置 **forwardHeaderPolicy**，您可以指定 Ingress 控制器何时和如何设置 **Forwarded**、**X-Forwarded-For**、**X-Forwarded-Host**、**X-Forwarded-Port**、**X-Forwarded-Proto** 和 **X-Forwarded-Proto-Version** HTTP 标头。 默认情况下，策略设置为 **Append**。 ● **Append** 指定 Ingress Controller 会附加标头，并保留任何现有的标头。 ● **Replace** 指定 Ingress Controller 设置标头，删除任何现有的标头。 ● **IfNone** 指定 Ingress Controller 在尚未设置标头时设置它们。 ● **Never** 指定 Ingress Controller 不会设置标头，并保留任何现有的标头。 通过设置 **headerNameCaseAdjustments**，您可以指定 HTTP 标头名对大小写的调整。每个调整都指定一个 HTTP 标头名称需要进行相关的大小写调整。例如，指定 **X-Forwarded-For** 表示 **x-forwarded-for** HTTP 标头应调整相应的大写。 这些调整仅应用于明文、边缘终止和重新加密路由，且仅在使用 HTTP/1 时有效。 对于请求标头，这些调整仅适用于具有 **haproxy.router.openshift.io/h1-adjust-case=true** 注解的路由。对于响应标头，这些调整适用于所有 HTTP 响应。如果此字段为空，则不会调整任何请求标头。 **actions** 指定对标头执行某些操作的选项。无法为 TLS 透传连接设置或删除标头。**actions** 字段具有额外的子字段 **spec.httpHeader.actions.response** 和 **spec.httpHeader.actions.request** ： ● **response** 子字段指定要设置或删除的 HTTP 响应标头列表。 ● **request** 子字段指定要设置或删除的 HTTP 请求标头列表。
httpCompression	**httpCompression** 定义 HTTP 流量压缩的策略。 ● **mimeTypes** 定义应该将压缩应用到的 MIME 类型列表。例如，**text/css; charset=utf-8**, **text/html**, **text/***, **image/svg+xml**, **application/octet-stream**, **X-custom/customsub**，格式为 **type/subtype; [;attribute=value]**。**types** 是：application, image, message, multipart, text, video, 或一个自定义类型（前面带有一个 **X-**；如需更详细的 MIME 类型和子类型的信息，请参阅 RFC1341
httpErrorCodePages	**httpErrorCodePages** 指定自定义 HTTP 错误代码响应页面。默认情况下，IngressController 使用 IngressController 镜像内构建的错误页面。

参数	描述
httpCaptureCookies	**httpCaptureCookies** 指定您要在访问日志中捕获的 HTTP cookie。如果 **httpCaptureCookies** 字段为空，则访问日志不会捕获 Cookie。 对于您要捕获的任何 Cookie，以下参数必须位于 **IngressController** 配置中： ● **name** 指定 Cookie 的名称。 ● **MaxLength** 指定 Cookie 的最大长度。 ● **matchType** 指定 Cookie 的 **name** 字段是否与捕获 Cookie 设置完全匹配，或者是捕获 Cookie 设置的前缀。**matchType** 字段使用 **Exact** 和 **Prefix** 参数。 例如： httpCaptureCookies: - matchType: Exact maxLength: 128 name: MYCOOKIE
httpCaptureHeaders	**httpCaptureHeaders** 指定要在访问日志中捕获的 HTTP 标头。如果 **httpCaptureHeaders** 字段为空，则访问日志不会捕获标头。 **httpCaptureHeaders** 包含两个要在访问日志中捕获的标头列表。这两个标题字段列表是 **request** 和 **response**。在这两个列表中，**name** 字段必须指定标头名称和 **maxlength** 字段，必须指定标头的最大长度。例如： httpCaptureHeaders: request: - maxLength: 256 name: Connection - maxLength: 128 name: User-Agent response: - maxLength: 256 name: Content-Type - maxLength: 256 name: Content-Length
tuningOptions	**tuningOptions** 指定用于调整 Ingress Controller pod 性能的选项。 ● **clientFinTimeout** 指定连接在等待客户端响应关闭连接时保持打开的时长。默认超时为 **1s**。 ● **clientTimeout** 指定连接在等待客户端响应时保持打开的时长。默认超时为 **30s**。 ● **headerBufferBytes** 为 Ingress Controller 连接会话指定保留多少内存（以字节为单位）。如果为 Ingress Controller 启用了 HTTP/2，则必须至少为 **16384**。如果没有设置，则默认值为 **32768** 字节。不建议设置此字段，因为 **headerBufferBytes** 值太小可能会破坏 Ingress Controller，而 **headerBufferBytes** 值过大可能会导致 Ingress Controller 使用比必要多的内存。 ● **headerBufferMaxRewriteBytes** 指定从 **headerBufferBytes** 为 Ingress Controller 连接会话保留多少内存（以字节为单位），用于

参数	描述

	HTTP 标头重写和附加。**headerBufferMaxRewriteBytes** 的最小值是 **4096**。**headerBufferBytes** 必须大于 **headerBufferMaxRewriteBytes**，用于传入的 HTTP 请求。如果没有设置，则默认值为 **8192** 字节。不建议设置此字段，因为 **headerBufferMaxRewriteBytes** 值可能会破坏 Ingress Controller，**headerBufferMaxRewriteBytes** 值太大可能会导致 Ingress Controller 使用比必要大得多的内存。
	
	• **healthCheckInterval** 指定路由器在健康检查之间等待的时间。默认值为 **5s**。
	
	• **serverFinTimeout** 指定连接在等待服务器响应关闭连接时保持打开的时长。默认超时为 **1s**。
	
	• **serverTimeout** 指定连接在等待服务器响应时保持打开的时长。默认超时为 **30s**。
	
	• **threadCount** 指定每个 HAProxy 进程创建的线程数量。创建更多线程可让每个 Ingress Controller pod 处理更多连接，而代价会增加所使用的系统资源。HAProxy 支持多达 **64** 个线程。如果此字段为空，Ingress Controller 将使用默认值 **4** 个线程。默认值可能会在以后的版本中改变。不建议设置此字段，因为增加 HAProxy 线程数量可让 Ingress Controller pod 在负载下使用更多 CPU 时间，并阻止其他 pod 收到需要执行的 CPU 资源。减少线程数量可能会导致 Ingress Controller 执行不佳。
	
	• **tlsInspectDelay** 指定路由器可以保存数据以查找匹配的路由的时长。如果把这个值设置得太短，对于 edge-terminated, reencrypted, 或 passthrough 的路由，则可能会导致路由器回退到使用默认证书，即使正在使用一个更加匹配的证书时也是如此。默认检查延迟为 **5s**。
	
	• **tunnelTimeout** 指定隧道连接在隧道闲置期间保持打开的时长，包括 websockets。默认超时为 **1h**。
	
	• **maxConnections** 指定每个 HAProxy 进程可建立的最大同时连接数。增加这个值可让每个入口控制器 pod 以额外的系统资源成本处理更多连接。允许的值是 **0**、**-1**、以及范围为 **2000** 和 **2000000** 内的任何值，或者字段可以留空。
	
	  ◦ 如果此字段留空，或者值为 **0**，Ingress Controller 将使用默认值 **50000**。这个值可能在以后的版本中有所改变。
	
	  ◦ 如果字段的值为 **-1**，则 HAProxy 将根据运行中容器中的可用 **ulimits** 动态计算最大值。与当前默认值 **50000** 相比，此进程会产生很大的内存用量。
	
	  ◦ 如果字段的值大于当前操作系统的限制，则 HAProxy 进程将不会启动。
	
	  ◦ 如果您选择了一个离散值，并且路由器 pod 迁移到新节点，则新节点可能没有配置相同的 **ulimit**。在这种情况下，pod 无法启动。
	
	  ◦ 如果您配置了不同的 **ulimits** 的节点，并且您选择离散值，则建议为该字段使用 **-1** 的值，以便在运行时计算连接的最大数量。

参数	描述
logEmptyRequests	**logEmptyRequests** 指定没有接收和记录请求的连接。这些空请求来自负载均衡器健康探测或 Web 浏览器规范连接(preconnect)，并记录这些请求。但是，这些请求可能是由网络错误导致的，在这种情况下，记录空请求可用于诊断错误。这些请求可能是由端口扫描导致的，记录空请求有助于检测入侵尝试。此字段允许的值有 **Log** 和 **Ignore**。默认值为 **Log**。 **LoggingPolicy** 类型接受以下两个值之一： ● **log** ：将此值设置为 **Log** 表示应记录某一事件。 ● **ignore** ：将此值设置为 **Ignore** 会在 HAproxy 配置中设置 **dontlognull** 选项。
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy** 描述了在收到请求前发生超时时，如何处理 HTTP 连接。此字段允许的值是 **Respond** 和 **Ignore**。默认值为 **Respond**。 **HTTPEmptyRequestsPolicy** 类型接受以下两个值之一： ● **Respond**：如果字段设置为 **Respond**，Ingress Controller 会发送 HTTP **400** 或 **408** 响应，在启用了访问日志时记录连接，并在适当的指标中计数连接。 ● **ignore**：将这个选项设置为 **Ignore** 会在 HAproxy 配置中添加 **http-ignore-probes** 参数。如果字段设置为 **Ignore**，Ingnore 会在不发送响应的情况下关闭连接，然后记录连接或递增指标。 这些连接来自负载均衡器健康探测或 Web 浏览器规范连接（预连接），可以安全地忽略。但是，这些请求可能是由网络错误造成的，因此将此字段设置为 **Ignore** 可能会妨碍对问题的检测和诊断。这些请求可能是由端口扫描导致的，在这种情况下，记录空请求有助于检测入侵尝试。

2.2.3.1. Ingress Controller TLS 安全配置集

TLS 安全配置文件为服务器提供了一种方式，以规范连接的客户端在连接服务器时可以使用哪些密码。

2.2.3.1.1. 了解 TLS 安全配置集

您可以使用 TLS （传输层安全）安全配置集来定义各种 OpenShift Dedicated 组件需要哪些 TLS 密码。OpenShift Dedicated TLS 安全配置集基于 Mozilla 推荐的配置。

您可以为每个组件指定以下 TLS 安全配置集之一：

表 2.1. TLS 安全配置集

profile	描述

profile	描述
Old	此配置集用于旧的客户端或库。该配置集基于旧的向后兼容性建议配置。 **Old** 配置集要求最低 TLS 版本 1.0。  **注意** 对于 Ingress Controller，最小 TLS 版本从 1.0 转换为 1.1。
Intermediate	这个配置集是大多数客户端的建议配置。它是 Ingress Controller、kubelet 和 control plane 的默认 TLS 安全配置集。该配置集基于 Intermediate 兼容性推荐的配置。 **Intermediate** 配置集需要最小 TLS 版本 1.2。
Modern	此配置集主要用于不需要向后兼容的现代客户端。这个配置集基于 Modern 兼容性推荐的配置。 **Modern** 配置集需要最低 TLS 版本 1.3。
Custom	此配置集允许您定义要使用的 TLS 版本和密码。  **警告** 使用 **Custom** 配置集时要谨慎，因为无效的配置可能会导致问题。

注意

当使用预定义的配置集类型时，有效的配置集配置可能会在发行版本之间有所改变。例如，使用在版本 X.Y.Z 中部署的 Intermediate 配置集指定了一个规格，升级到版本 X.Y.Z+1 可能会导致应用新的配置集配置，从而导致推出部署。

2.2.3.1.2. 为 Ingress Controller 配置 TLS 安全配置集

要为 Ingress Controller 配置 TLS 安全配置集，请编辑 **IngressController** 自定义资源（CR）来指定预定义或自定义 TLS 安全配置集。如果没有配置 TLS 安全配置集，则默认值基于为 API 服务器设置的 TLS 安全配置集。

配置 Old TLS 安全配置集的 IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
```

```
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS 安全配置集定义 Ingress Controller 的 TLS 连接的最低 TLS 版本和 TLS 密码。

您可以在 **Status.Tls Profile** 和 **Spec.Tls Security Profile** 下看到 **IngressController** 自定义资源（CR）中配置的 TLS 安全配置集的密码和最小 TLS 版本。对于 **Custom** TLS 安全配置集，这两个参数下列出了特定的密码和最低 TLS 版本。

注意

HAProxy Ingress Controller 镜像支持 TLS **1.3** 和 **Modern** 配置集。

Ingress Operator 还会将 **Old** 或 **Custom** 配置集的 TLS **1.0** 转换为 **1.1**。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 编辑 **openshift-ingress-operator** 项目中的 **IngressController** CR，以配置 TLS 安全配置集：

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

2. 添加 **spec.tlsSecurityProfile** 字段：

Custom 配置集的 IngressController CR 示例

```
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) 指定 TLS 安全配置集类型（**Old**、**Intermediate** 或 **Custom**）。默认值为 **Intermediate**。

(2) 为所选类型指定适当的字段：

- **old: {}**

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

3 对于 **custom** 类型，请指定 TLS 密码列表和最低接受的 TLS 版本。

3. 保存文件以使改变生效。

验证

- 验证 **IngressController** CR 中是否设置了配置集：

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

输出示例

```
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
 ...
```

2.2.3.1.3. 配置 mutual TLS 身份验证

您可以通过设置 **spec.clientTLS** 值，将 Ingress Controller 配置为启用 mutual TLS (mTLS) 身份验证。**clientTLS** 值将 Ingress Controller 配置为验证客户端证书。此配置包括设置 **clientCA** 值，这是对配置映射的引用。配置映射包含 PEM 编码的 CA 证书捆绑包，用于验证客户端的证书。另外，您还可以配置证书主题过滤器列表。

如果 **clientCA** 值指定了 X509v3 证书撤销列表 (CRL) 分发点，Ingress Operator 会下载并管理基于每个提供的证书中指定的 HTTP URI X509v3 **CRL 分发点**的 CRL 配置映射。Ingress Controller 在 mTLS/TLS 协商过程中使用此配置映射。不提供有效证书的请求将被拒绝。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 您有一个 PEM 编码的 CA 证书捆绑包。

- 如果您的 CA 捆绑包引用 CRL 发布点，还必须将最终用户或叶证书包含在客户端 CA 捆绑包中。此证书必须在 **CRL 分发点**下包含 HTTP URI，如 RFC 5280 所述。例如：

```
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
```

流程

1. 在 **openshift-config** 命名空间中，从 CA 捆绑包创建配置映射：

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

(1) 配置映射数据键必须是 **ca-bundle.pem**，数据值必须是 PEM 格式的 CA 证书。

2. 编辑 **openshift-ingress-operator** 项目中的 **IngressController** 资源：

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

3. 添加 **spec.clientTLS** 字段和子字段来配置 mutual TLS：

指定过滤模式的 clientTLS 配置集的 IngressController CR 示例

```
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. 可选，输入以下命令获取 **允许的SubjectPatterns** 的可辨识名称(DN)。

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

2.2.4. 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 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
```

2.2.5. 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
```

2.2.6. 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>
```

2.2.7. 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>
```

2.2.8. 创建自定义 Ingress Controller

作为集群管理员，您可以创建新的自定义 Ingress Controller。因为默认 Ingress Controller 在 OpenShift Dedicated 更新过程中可能会有所变化，因此当维护一个在集群更新过程中保留的配置时，创建自定义 Ingress Controller 非常有用。

这个示例为自定义 Ingress Controller 提供最小规格。要进一步自定义自定义 Ingress Controller，请参阅"配置 Ingress Controller"。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建定义自定义 **IngressController** 对象的 YAML 文件：

custom-ingress-controller.yaml 文件示例

```
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 指定 **IngressController** 对象的自定义名称。

2 **使用自定义通配符证书指定 secret 名称。**

3 **最小副本需要是 ONE**

4 **指定您的域名的域。IngressController 对象中指定的域以及用于证书的域必须匹配。例如，如果 domain 值为 "custom_domain.mycompany.com"，则证书必须具有 SAN. <custom_domain.mycompany.com （在向域中添加.com）。**

2. **运行以下命令来创建对象：**

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

2.2.9. 配置 Ingress Controller

2.2.9.1. 设置自定义默认证书

作为管理员，您可以通过创建 Secret 资源并编辑 IngressController 自定义资源 (CR)，将 Ingress Controller 配置为使用自定义证书。

先决条件

- 您必须在 PEM 编码文件中有一个证书/密钥对，其中该证书由可信证书认证机构签名，或者由您在一个自定义 PKI 中配置的私有可信证书认证机构签名。

- 您的证书满足以下要求：

  - 该证书对入口域有效。

  - 证书使用 subjectAltName 扩展来指定通配符域，如 *.apps.ocp4.example.com。

- 您必须有一个 IngressController CR。您可以使用默认值：

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

输出示例

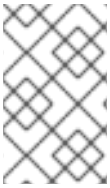
```
NAME     AGE
default  10m
```

注意

如果您有中间证书，则必须将其包含在包含自定义默认证书的 secret 的 tls.crt 文件中。指定证书时指定的顺序是相关的; 在任意服务器证书后列出您的中间证书。

流程

以下步骤假定自定义证书和密钥对位于当前工作目录下的 tls.crt 和 tls.key 文件中。替换 tls.crt 和 tls.key 的实际路径名。在创建 Secret 资源并在 IngressController CR 中引用它时，您也可以将 custom-certs-default 替换成另一名称。



注意

此操作会导致使用滚动部署策略重新部署 Ingress Controller。

1. 使用 tls.crt 和 tls.key 文件，创建在 openshift-ingress 命名空间中包含自定义证书的 Secret 资源。

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

2. 更新 IngressController CR，以引用新的证书 Secret：

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

3. 验证更新是否已生效：

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

其中：

<domain>

指定集群的基域名。

输出示例

```
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
```

提示

您还可以应用以下 YAML 来设置自定义默认证书：

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

证书 Secret 名称应该与用来更新 CR 的值匹配。

修改了 IngressController CR 后，Ingress Operator 将更新 Ingress Controller 的部署以使用自定义证书。

2.2.9.2. 删除自定义默认证书

作为管理员，您可以删除配置了 Ingress Controller 的自定义证书。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 已安装 OpenShift CLI(oc)。
- 您之前为 Ingress Controller 配置了自定义默认证书。

流程

- 要删除自定义证书并恢复 OpenShift Dedicated 附带的证书，请输入以下命令：

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

集群协调新证书配置时可能会有延迟。

验证

- 要确认原始集群证书已被恢复，请输入以下命令：

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

其中：

<domain>

指定集群的基域名。

输出示例

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

2.2.9.3. 自动扩展 Ingress Controller

您可以自动扩展 Ingress Controller 以动态满足路由性能或可用性要求，如提高吞吐量的要求。

以下流程提供了扩展默认 Ingress Controller 的示例。

先决条件

- 已安装 OpenShift CLI (oc)。

- 您可以使用具有 cluster-admin 角色的用户访问 OpenShift Dedicated 集群。
- 已安装自定义 Metrics Autoscaler Operator 和关联的 KEDA Controller。
  - 您可以在 Web 控制台中使用 OperatorHub 安装 Operator。安装 Operator 后，您可以创建 KedaController 实例。

流程

1. 运行以下命令，创建一个服务帐户来与 Thanos 进行身份验证：

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

输出示例

```
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. 使用以下命令手动创建服务帐户令牌：

```
$ 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.

使用服务帐户的令牌，在 openshift-ingress-operator 命名空间中定义一个 TriggerAuthentication 对象。

a. 创建 TriggerAuthentication 对象，并将 secret 变量的值传递给 TOKEN 参数：

```
$ 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. 创建并应用角色以从 Thanos 读取指标：

a. 创建一个新角色 thanos-metrics-reader.yaml，从 pod 和节点读取指标：

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. 运行以下命令来应用新角色：

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

5. 输入以下命令在服务帐户中添加新角色：

```
$ 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
```

注意

只有在使用跨命名空间查询时，才需要参数 add-cluster-role-to-user。以下步骤使用 kube-metrics 命名空间中的查询，该命名空间需要此参数。

6. 创建一个新的 ScaledObject YAML 文件 ingress-autoscaler.yaml，该文件以默认 Ingress Controller 部署为目标：

ScaledObject 定义示例

```
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) 要目标的自定义资源。在本例中，Ingress Controller。

(2) 可选：最大副本数。如果省略此字段，则默认最大值设置为 100 个副本。

(3) openshift-monitoring 命名空间中的 Thanos 服务端点。

(4) Ingress Operator 命名空间。

(5) 此表达式评估为，部署的集群中存在很多 worker 节点。

重要

如果使用跨命名空间查询，您必须在 serverAddress 字段中目标端口 9091 而不是端口 9092。您还必须有升级的特权，才能从此端口读取指标。

7. 运行以下命令来应用自定义资源定义：

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

验证

- 运行以下命令，验证默认 Ingress Controller 是否已扩展以匹配 kube-state-metrics 查询返回的值：
  - 使用 grep 命令搜索 Ingress Controller YAML 文件以查找副本：

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

输出示例

```
  replicas: 3
```

  - 获取 openshift-ingress 项目中的 pod：

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

输出示例

```
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
```

2.2.9.4. 扩展 Ingress Controller

手动扩展 Ingress Controller 以满足路由性能或可用性要求，如提高吞吐量的要求。oc 命令用于扩展 IngressController 资源。以下流程提供了扩展默认 IngressController 的示例。

注意

扩展不是立刻就可以完成的操作，因为它需要时间来创建所需的副本数。

流程

1. 查看默认 IngressController 的当前可用副本数：

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

输出示例

```
2
```

2. 使用 oc patch 命令，将默认 IngressController 扩展至所需的副本数。以下示例将默认 IngressController 扩展至 3 个副本：

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

输出示例

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

3. 验证默认 IngressController 是否已扩展至您指定的副本数：

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

输出示例

```
3
```

提示

您还可以应用以下 YAML 将 Ingress Controller 扩展为三个副本：

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

1 如果需要不同数量的副本，请更改 replicas 值。

2.2.9.5. 配置 Ingress 访问日志

您可以配置 Ingress Controller 以启用访问日志。如果您的集群没有接收许多流量，那么您可以将日志记录到 sidecar。如果您的集群接收大量流量，为了避免超出日志记录堆栈的容量，或与 OpenShift Dedicated 之外的日志记录基础架构集成，您可以将日志转发到自定义 syslog 端点。您还可以指定访问日志的格式。

当不存在 Syslog 日志记录基础架构时，容器日志记录可用于在低流量集群中启用访问日志，或者在诊断 Ingress Controller 时进行简短使用。

对于访问日志可能会超过 OpenShift Logging 堆栈容量的高流量集群，或需要任何日志记录解决方案与现有 Syslog 日志记录基础架构集成的环境，则需要 syslog。Syslog 用例可能会相互重叠。

先决条件

- 以具有 cluster-admin 特权的用户身份登录。

流程

配置 Ingress 访问日志到 sidecar。

- 要配置 Ingress 访问日志记录，您必须使用 spec.logging.access.destination 指定一个目的地。要将日志记录指定到 sidecar 容器，您必须指定 Container spec.logging.access.destination.type。以下示例是将日志记录到 Container 目的地的 Ingress Controller 定义：

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

- 当将 Ingress Controller 配置为日志记录到 sidecar 时，Operator 会在 Ingress Controller Pod 中创建一个名为 logs 的容器：

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

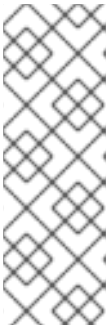
输出示例

```
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"
```

配置 Ingress 访问日志记录到 Syslog 端点。

- 要配置 Ingress 访问日志记录，您必须使用 spec.logging.access.destination 指定一个目的地。要将日志记录指定到 Syslog 端点目的地，您必须为 spec.logging.access.destination.type 指定 Syslog。如果目的地类型是 Syslog，还必须使用 spec.logging.access.destination.syslog.address 指定目标端点，您可以使用 spec.logging.access.destination.syslog.facility 指定一个工具。以下示例是将日志记录到 Syslog 目的地的 Ingress Controller 定义：

```
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
```



注意

Syslog 目的地端口必须是 UDP。

syslog 目标地址必须是 IP 地址。它不支持 DNS 主机名。

使用特定的日志格式配置 Ingress 访问日志。

- 您可以指定 spec.logging.access.httpLogFormat 来自定义日志格式。以下示例是一个 Ingress Controller 定义，它将日志记录到 IP 地址为 1.2.3.4、端口为 10514 的 syslog 端点：

```
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'
```

禁用 Ingress 访问日志。

- 要禁用 Ingress 访问日志，请保留 spec.logging 或 spec.logging.access 为空：

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

允许 Ingress Controller 在使用 sidecar 时，修改 HAProxy 日志长度。

- 如果您使用 spec.logging.access.destination.syslog.maxLength，请使用 spec.logging.access.destination.type: Syslog。

```
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
```

- 如果您使用 spec.logging.access.destination.container.maxLength，请使用 spec.logging.access.destination.type: Container。

```
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
```

2.2.9.6. 设置 Ingress Controller 线程数

集群管理员可设置线程数，以增加集群可以处理的入站的连接量。您可以修补现有的 Ingress Controller 来增加线程量。

先决条件

- 以下假设您已创建了 Ingress Controller。

流程

- 更新 Ingress Controller 以增加线程数量：

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

注意

如果您的节点有能力运行大量资源，您可以使用与预期节点容量匹配的标签配置 spec.nodePlacement.nodeSelector，并将 spec.tuningOptions.threadCount 配置为一个适当的高值。

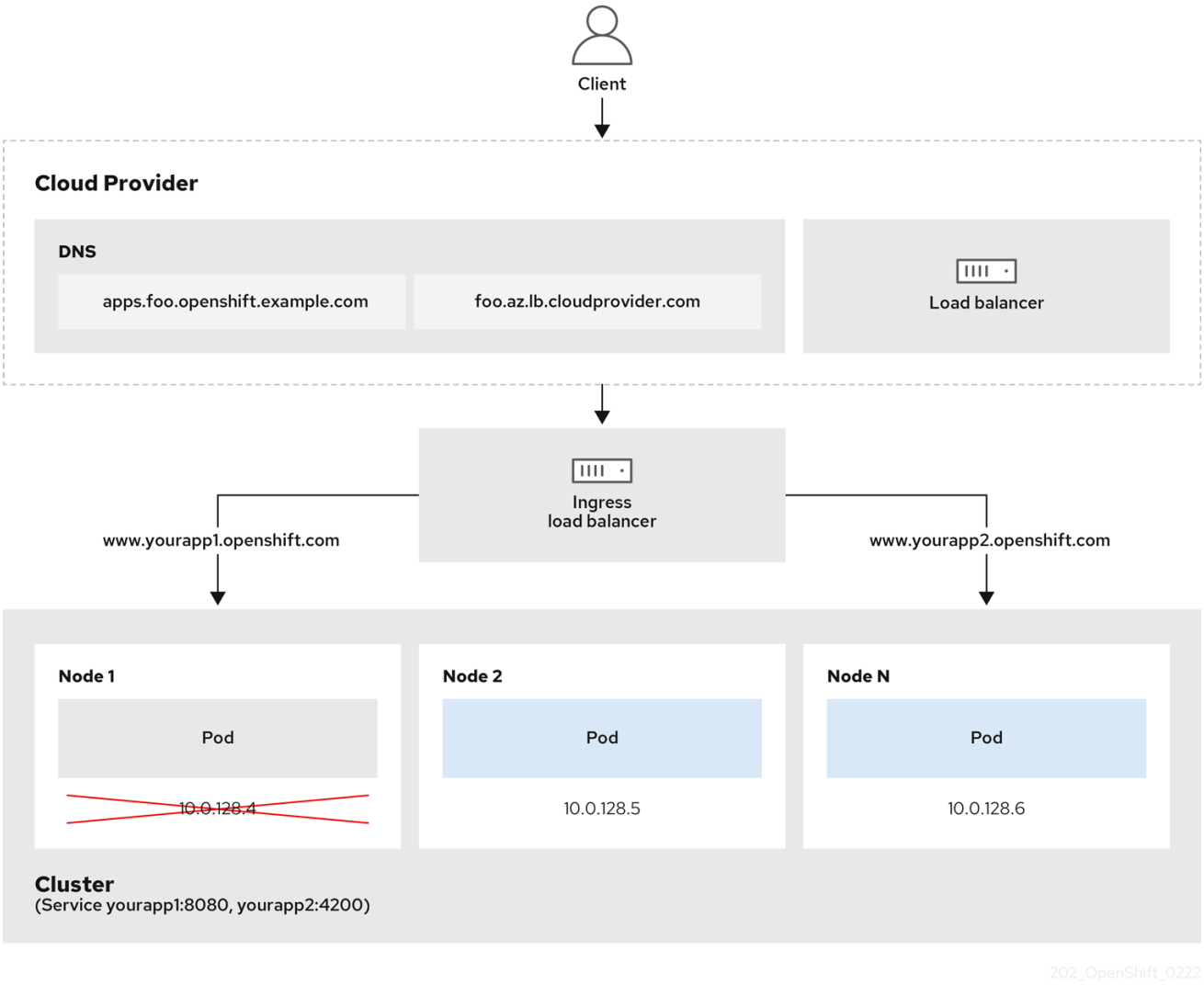
2.2.9.7. 配置 Ingress Controller 以使用内部负载均衡器

当在云平台上创建 Ingress Controller 时，Ingress Controller 默认由一个公共云负载均衡器发布。作为管理员，您可以创建一个使用内部云负载均衡器的 Ingress Controller。

重要

如果要更改 IngressController 的 scope，您可以在创建自定义资源(CR)后更改 .spec.endpointPublishingStrategy.loadBalancer.scope 参数。

图 2.1. LoadBalancer 图表



上图显示了与 OpenShift Dedicated Ingress LoadBalancerService 端点发布策略相关的以下概念：

- 您可以使用 OpenShift Ingress Controller Load Balancer 在外部使用云供应商负载均衡器或内部加载负载。
- 您可以使用负载均衡器的单个 IP 地址以及更熟悉的端口，如 8080 和 4200，如图形中所述的集群所示。

- 来自外部负载均衡器的流量定向到 pod，并由负载均衡器管理，如下节点的实例中所述。有关实现详情请查看 Kubernetes 服务文档 。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 在名为 <name>-ingress-controller.yaml 的文件中创建 IngressController 自定义资源 (CR)，如下例所示：

```
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. 将 <name> 替换为 IngressController 对象的名称。
2. 指定控制器发布的应用程序的 domain。
3. 指定一个 Internal 值以使用内部负载均衡器。

2. 运行以下命令，创建上一步中定义的 Ingress Controller：

-

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

1

将 <name> 替换为 IngressController 对象的名称。

3. 可选：通过运行以下命令确认创建了 Ingress Controller：

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

2.2.9.8. 设置 Ingress Controller 健康检查间隔

集群管理员可以设置健康检查间隔，以定义路由器在两个连续健康检查之间等待的时间。这个值会作为所有路由的默认值进行全局应用。默认值为 5 秒。

先决条件

- 以下假设您已创建了 Ingress Controller。

流程

- 更新 Ingress Controller，以更改后端健康检查之间的间隔：

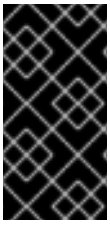
```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p
'{"spec":{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```

注意

要覆盖单个路由的 healthCheckInterval，请使用路由注解 router.openshift.io/haproxy.health.check.interval

2.2.9.9. 将集群的默认 Ingress Controller 配置为内部

您可以通过删除并重新它来将默认 Ingress Controller 配置为内部。



重要

如果要更改 IngressController 的 scope，您可以在创建自定义资源(CR)后更改 .spec.endpointPublishingStrategy.loadBalancer.scope 参数。

先决条件

- 安装 OpenShift CLI（oc）。
- 以具有 cluster-admin 特权的用户身份登录。

流程

1. 通过删除并重新创建集群，将 默认 Ingress Controller 配置为内部。

```
$ 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
```

2.2.9.10. 配置路由准入策略

管理员和应用程序开发人员可在多个命名空间中运行具有相同域名的应用程序。这是针对多个团队开发的、在同一个主机名上公开的微服务的机构。

警告

只有在命名空间间有信任的集群才会启用跨命名空间之间的声明，否则恶意用户可能会接管主机名。因此，默认的准入策略不允许在命名空间间声明主机名。

先决条件

- 必须具有集群管理员权限。

流程

- 使用以下命令编辑 ingresscontroller 资源变量的.spec. routeAdmission 字段：

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

Ingress 控制器配置参数

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

提示

您还可以应用以下 YAML 来配置路由准入策略：

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

2.2.9.11. 使用通配符路由

HAProxy Ingress Controller 支持通配符路由。Ingress Operator 使用 wildcardPolicy 来配置 Ingress Controller 的 ROUTER_ALLOW_WILDCARD_ROUTES 环境变量。

Ingress Controller 的默认行为是接受采用 None 通配符策略的路由，该策略与现有

IngressController 资源向后兼容。

流程

1. 配置通配符策略。

a. 使用以下命令来编辑 IngressController 资源：

```
$ oc edit IngressController
```

b. 在 spec 下，将 wildcardPolicy 字段设置 为 WildcardsDisallowed 或 WildcardsAllowed：

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

2.2.9.12. HTTP 标头配置

OpenShift Dedicated 提供了不同的使用 HTTP 标头的方法。在设置或删除标头时，您可以使用 Ingress Controller 中的特定字段或单独的路由来修改请求和响应标头。您还可以使用路由注解设置某些标头。配置标头的各种方法在协同工作时可能会带来挑战。

注意

您只能在 IngressController 或 Route CR 中设置或删除标头，您无法附加它们。如果使用值设置 HTTP 标头，则该值必须已完成，且在以后不需要附加。在附加标头（如 X-Forwarded-For 标头）时，请使用 spec.httpHeaders.forwardedHeaderPolicy 字段，而不是 spec.httpHeaders.actions。

2.2.9.12.1. 优先级顺序

当在 Ingress Controller 和路由中修改相同的 HTTP 标头时，HAProxy 会根据它是请求还是响应标头来优先选择操作。

- 对于 HTTP 响应标头，Ingress Controller 中指定的操作会在路由中指定的操作后执行。这意味着 Ingress Controller 中指定的操作具有优先权。

对于 HTTP 请求标头，路由中指定的操作会在 Ingress Controller 中指定的操作后执行。这意味着路由中指定的操作具有优先权。

例如，集群管理员使用以下配置设置 X-Frame-Options 响应标头，其值为 DENY ：

IngressController spec 示例

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

路由所有者设置 Ingress Controller 中设置的相同响应标头，但使用以下配置值 SAMEORIGIN ：

Route 规格示例

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

当 IngressController spec 和 Route spec 都配置 X-Frame-Options 响应标头时，Ingress Controller 的全局级别上为此标头设置的值具有优先权，即使一个特定的路由允许帧。对于请求标头，Route spec 值会覆盖 IngressController spec 值。

这是因为 haproxy.config 文件使用以下逻辑，其中 Ingress Controller 被视为前端，单个路由被视为后端。应用到前端配置的标头值 DENY 使用后端中设置的值 SAMEORIGIN 覆盖相同的标头：

```
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'
```

另外，Ingress Controller 或路由中定义的任何操作都覆盖使用路由注解设置的值。

2.2.9.12.2. 特殊情况标头

以下标头可能会阻止完全被设置或删除，或者在特定情况下允许：

表 2.2. 特殊情况标头配置选项

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
proxy	否	否	proxy HTTP 请求标头可以通过将标头值注入 HTTP_PROXY 环境变量来利用这个安全漏洞的 CGI 应用程序。proxy HTTP 请求标头也是非标准的，在配置期间容易出错。	否

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
主机	否	是	当使用 **IngressController** CR 设置 **host** HTTP 请求标头时，HAProxy 在查找正确的路由时可能会失败。	否
strict-transport-security	否	否	**strict-transport-security** HTTP 响应标头已使用路由注解处理，不需要单独的实现。	是：**haproxy.router.openshift.io/hsts_header** 路由注解
cookie 和 **set-cookie**	否	否	HAProxy 集的 Cookie 用于会话跟踪，用于将客户端连接映射到特定的后端服务器。允许设置这些标头可能会影响 HAProxy 的会话关联，并限制 HAProxy 的 Cookie 的所有权。	是： • **haproxy.router.openshift.io/disable_cookie** 路由注解 • **haproxy.router.openshift.io/cookie_name** 路由注解

2.2.9.13. 在 Ingress Controller 中设置或删除 HTTP 请求和响应标头

出于合规的原因，您可以设置或删除某些 HTTP 请求和响应标头。您可以为 Ingress Controller 提供的所有路由或特定路由设置或删除这些标头。

例如，您可能希望将集群中运行的应用程序迁移到使用 mutual TLS，这需要您的应用程序检查 X-Forwarded-Client-Cert 请求标头，但 OpenShift Dedicated 默认 Ingress Controller 提供了一个 X-SSL-Client-Der 请求标头。

以下流程修改 Ingress Controller 来设置 X-Forwarded-Client-Cert 请求标头，并删除 X-SSL-Client-Der 请求标头。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问 OpenShift Dedicated 集群。

流程

1. 编辑 Ingress Controller 资源：

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

2. 将 X-SSL-Client-Der HTTP 请求标头替换为 X-Forwarded-Client-Cert HTTP 请求标头：

```
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
```

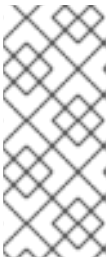
1. 要在 HTTP 标头上执行的操作列表。
2. 您要更改的标头类型。在本例中，请求标头。
3. 您要更改的标头的名称。有关您可以设置或删除的可用标头列表，请参阅 *HTTP 标头配置*。

4

在标头中执行的操作类型。此字段可以具有 Set 或 Delete 的值。

5

在设置 HTTP 标头时，您必须提供一个 value。该值可以是该标头的可用指令列表中的字符串，如 DENY，也可以是使用 HAProxy 的动态值语法来解释的动态值。在这种情况下，会添加一个动态值。



注意

对于 HTTP 响应设置动态标头值，允许示例 fetchers 是 res.hdr 和 ssl_c_der。对于 HTTP 请求设置动态标头值，允许示例获取器为 req.hdr 和 ssl_c_der。请求和响应动态值都可以使用 lower 和 base64 转换器。

3. 保存文件以使改变生效。

2.2.9.14. 使用 X-Forwarded 标头

您可以将 HAProxy Ingress Controller 配置为指定如何处理 HTTP 标头的策略，其中包括 Forwarded 和 X-Forwarded-For。Ingress Operator 使用 HTTPHeaders 字段配置 Ingress Controller 的 ROUTER_SET_FORWARDED_HEADERS 环境变量。

流程

1. 为 Ingress Controller 配置 HTTPHeaders 字段。

   a. 使用以下命令来编辑 IngressController 资源：

   ```
   $ oc edit IngressController
   ```

   b. 在 spec 下，将 HTTPHeaders 策略字段设置为 Append、Replace、IfNone 或 Never：

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

```
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

使用案例示例

作为集群管理员，您可以：

- 配置将 X-Forwarded-For 标头注入每个请求的外部代理，然后将其转发到 Ingress Controller。

  要将 Ingress Controller 配置为通过未修改的标头传递，您需要指定 never 策略。然后，Ingress Controller 不会设置标头，应用程序只接收外部代理提供的标头。

- 将 Ingress Controller 配置为通过未修改的外部代理在外部集群请求上设置 X-Forwarded-For 标头。

  要将 Ingress Controller 配置为在不通过外部代理的内部集群请求上设置 X-Forwarded-For 标头，请指定 if-none 策略。如果 HTTP 请求已经通过外部代理设置了标头，则 Ingress Controller 会保留它。如果缺少标头，因为请求没有通过代理，Ingress Controller 会添加标头。

作为应用程序开发人员，您可以：

- 配置特定于应用程序的外部代理来注入 X-Forwarded-For 标头。

  要配置 Ingress Controller，以便在不影响其他路由策略的情况下将标头传递到应用程序的路由，请在应用程序的路由上添加注解 haproxy.router.openshift.io/set-forwarded-headers: if-none 或 haproxy.router.openshift.io/set-forwarded-headers: never。

注意

您可以根据每个路由设置 haproxy.router.openshift.io/set-forwarded-headers 注解，独立于 Ingress Controller 的全局设置值。

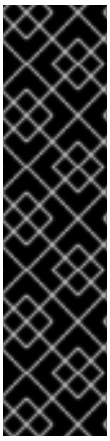
2.2.9.15. 启用 HTTP/2 入口连接

您可以在 HAProxy 中启用透明端到端的 HTTP/2 连接。此功能使应用程序所有者利用 HTTP/2 协议功能，包括单一连接、标头压缩、二 进制流等等。

您可以为单独的 Ingress Controller 或整个集群启用 HTTP/2 连接。

要在从客户端到 HAProxy 的连接中启用 HTTP/2，路由必须指定一个自定义证书。使用默认证书的路由无法使用 HTTP/2。这一限制是避免连接并发问题（如客户端为使用相同证书的不同路由重新使用连接）所必需的。

从 HAProxy 到应用程序 pod 的连接只能将 HTTP/2 用于 re-encrypt 路由，而不适用于 edge-terminated 或 insecure 路由。存在这个限制的原因是，在与后端协商使用 HTTP/2 时，HAProxy 要使用 ALPN（Application-Level Protocol Negotiation），它是一个 TLS 的扩展。这意味着，端到端的 HTTP/2 适用于 passthrough 和 re-encrypt 路由，而不适用于 nsecure 或 edge-terminated 路由。



重要

对于非 passthrough 路由，Ingress Controller 会独立于客户端的连接来协商它与应用程序的连接。这意味着，客户端可以连接到 Ingress Controller 并协商 HTTP/1.1，Ingress Controller 可连接到应用程序，协商 HTTP/2 并使用 HTTP/2 连接将客户端 HTTP/1.1 连接转发请求。如果客户端随后试图将其连接从 HTTP/1.1 升级到 WebSocket 协议，这会导致问题。因为 Ingress Controller 无法将 WebSocket 转发到 HTTP/2，也无法将其 HTTP/2 的连接升级到 WebSocket。因此，如果您有一个应用程序旨在接受 WebSocket 连接，则必须允许使用 HTTP/2 协议，或者其它客户端将无法升级到 WebSocket 协议。

流程

在单一 Ingress Controller 上启用 HTTP/2。

- 要在 Ingress Controller 上启用 HTTP/2，请输入 oc annotate 命令：

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

将 <ingresscontroller_name> 替换为要注解的 Ingress Controller 的名称。

在整个集群中启用 HTTP/2。

- 要为整个集群启用 HTTP/2，请输入 oc annotate 命令：

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

提示

您还可以应用以下 YAML 来添加注解：

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

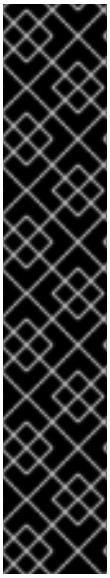
2.2.9.16. 为 Ingress Controller 配置 PROXY 协议

当 Ingress Controller 使用 HostNetwork、NodePortService 或 私有 端点发布策略类型时，集群管理员可以配置 PROXY 协议。PROXY 协议使负载均衡器能够为 Ingress Controller 接收的连接保留原始客户端地址。原始客户端地址可用于记录、过滤和注入 HTTP 标头。在默认配置中，Ingress Controller 接收的连接只包含与负载均衡器关联的源地址。

警告

使用 Keepalived Ingress Virtual IP (VIP)在非云平台上带有安装程序置备的集群的默认 Ingress Controller 不支持 PROXY 协议。

PROXY 协议使负载均衡器能够为 Ingress Controller 接收的连接保留原始客户端地址。原始客户端地址可用于记录、过滤和注入 HTTP 标头。在默认配置中，Ingress Controller 接收的连接只包含与负载均衡器关联的源 IP 地址。



重要

对于 passthrough 路由配置，OpenShift Dedicated 集群中的服务器无法观察原始客户端源 IP 地址。如果您需要知道原始客户端源 IP 地址，请为 Ingress Controller 配置 Ingress 访问日志记录，以便您可以查看客户端源 IP 地址。

对于重新加密和边缘路由，OpenShift Dedicated 路由器设置 Forwarded 和 X-Forwarded-For 标头，以便应用程序工作负载检查客户端源 IP 地址。

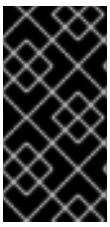
如需有关 Ingress 访问日志的更多信息，请参阅"配置 Ingress 访问日志"。

使用 LoadBalancerService 端点发布策略类型时不支持为 Ingress Controller 配置 PROXY 协议。这个限制是因为当 OpenShift Dedicated 在云平台中运行时，Ingress Controller 指定应使用服务负载均衡器，Ingress Operator 配置负载均衡器服务并根据保留源地址的平台要求启用 PROXY 协议。

重要

您必须将 OpenShift Dedicated 和外部负载均衡器配置为使用 PROXY 协议或 TCP。

云部署不支持此功能。这个限制是因为当 OpenShift Dedicated 在云平台中运行时，Ingress Controller 指定应使用服务负载均衡器，Ingress Operator 配置负载均衡器服务并根据保留源地址的平台要求启用 PROXY 协议。



重要

您必须将 OpenShift Dedicated 和外部负载均衡器配置为使用 PROXY 协议或使用传输控制协议(TCP)。

先决条件

- 已创建一个 Ingress Controller。

流程

1. 在 CLI 中输入以下命令来编辑 Ingress Controller 资源：

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

2. 设置 PROXY 配置：

- 如果您的 Ingress Controller 使用 HostNetwork 端点发布策略类型，请将 spec.endpointPublishingStrategy.hostNetwork.protocol 子字段设置为 PROXY ：

hostNetwork 配置为 PROXY 的示例

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

- 如果您的 Ingress Controller 使用 NodePortService 端点发布策略类型，请将 spec.endpointPublishingStrategy.nodePort.protocol 子字段设置为 PROXY ：

nodePort 配置为 PROXY 示例

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

- 如果您的 Ingress Controller 使用 Private 端点发布策略类型，请将 spec.endpointPublishingStrategy.private.protocol 子字段设置为 PROXY ：

到 PROXY 的私有 配置示例

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

其他资源

- 配置 Ingress 访问日志

2.2.9.17. 使用 appsDomain 选项指定备选集群域

作为集群管理员，您可以通过配置 appsDomain 字段来为用户创建的路由指定默认集群域替代内容。appsDomain 字段是 OpenShift Dedicated 使用的可选域，而不是默认值，默认值在 domain 字段中指定。如果您指定了其它域，它会覆盖为新路由确定默认主机的目的。

例如，您可以将您公司的 DNS 域用作集群中运行的应用程序的路由和入口的默认域。

先决条件

- 已部署了 OpenShift Dedicated 集群。
- 已安装 oc 命令行界面。

流程

1. 通过为用户创建的路由指定备选默认域来配置 appsDomain 字段。

   a. 编辑 ingress 集群资源 ：

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

b. 编辑 YAML 文件：

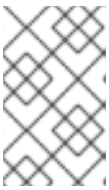
示例 appsDomain 配置为 test.example.com

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

(1) 指定默认域。您不能在安装后修改默认域。

(2) 可选：用于应用程序路由的 OpenShift Dedicated 基础架构域。您可以使用 测试 等替代前缀 apps，而不是默认前缀。

2. 通过公开路由并验证路由域更改，验证现有路由是否包含 appsDomain 字段中指定的域名：



注意

在公开路由前，等待 openshift-apiserver 完成滚动更新。

a. 公开路由：

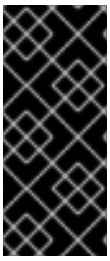
```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

输出示例

```
$ 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
```

2.2.9.18. 转换 HTTP 标头的大小写

默认情况下，HAProxy 小写 HTTP 标头名称；例如，将 Host: xyz.com 更改为 host: xyz.com。如果旧应用程序对 HTTP 标头名称中使用大小写敏感，请使用 Ingress Controller spec.httpHeaders.headerNameCaseAdjustments API 字段进行调整来适应旧的应用程序，直到它们被改变。



重要

OpenShift Dedicated 包括 HAProxy 2.8。如果要更新基于 web 的负载均衡器的这个版本，请确保将 spec.httpHeaders.headerNameCaseAdjustments 部分添加到集群的配置文件中。

作为集群管理员，您可以使用 oc patch 命令，或设置 Ingress Controller YAML 文件中的 HeaderNameCaseAdjustments 字段来转换 HTTP 标头的大小写。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

- 使用 oc patch 命令大写 HTTP 标头。

a. 运行以下命令，将 HTTP 标头 从主机 更改为 Host ：

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

b. 创建 Route 资源 YAML 文件，以便注解可应用到应用程序。

名为 my-application的路由示例

```
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) 设置 haproxy.router.openshift.io/h1-adjust-case，以便 Ingress Controller 能够调整 指定的主机 请求标头。

- 通过在 Ingress Controller YAML 配置文件中配置 HeaderNameCaseAdjustments 字段指定调整。

a. 以下示例 Ingress Controller YAML 文件将 HTTP/1 请求的 host 标头调整为 Host，以适当地注解路由：

Ingress Controller YAML 示例

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

b.

以下示例路由通过使用 haproxy.router.openshift.io/h1-adjust-case 注解启用 HTTP 响应标头名称大小调整：

路由 YAML 示例

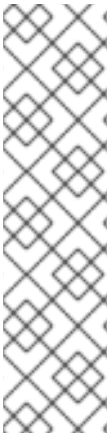
```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true 1
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

1

将 haproxy.router.openshift.io/h1-adjust-case 设置为 true。

2.2.9.19. 使用路由器压缩

您可以将 HAProxy Ingress Controller 配置为为特定 MIME 类型全局指定路由器压缩。您可以使用 mimeTypes 变量定义压缩应用到的 MIME 类型的格式。类型包括：application, image, message, multipart, text, video, 或带有一个 "X-" 前缀的自定义类型。要查看 MIME 类型和子类型的完整表示法，请参阅 RFC1341。



注意

为压缩分配的内存可能会影响最大连接。此外，对大型缓冲区的压缩可能导致延迟，如非常复杂的正则表达式或较长的正则表达式列表。

并非所有 MIME 类型从压缩中受益，但 HAProxy 仍然使用资源在指示时尝试压缩。通常而言，文本格式（如 html、css 和 js）与压缩格式获益，但已经压缩的格式（如图像、音频和视频）可能会因为需要压缩操作而无法获得太多的好处。

流程

1. 为 Ingress Controller 配置 httpCompression 字段。

   a. 使用以下命令来编辑 IngressController 资源：

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

   b. 在 spec 下，将 httpCompression 策略字段设置为 mimeTypes，并指定应该应用压缩的 MIME 类型列表：

   ```
   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"
       ...
   ```

2.2.9.20. 公开路由器指标

您可以在默认统计端口 1936 上以 Prometheus 格式公开 HAProxy 路由器指标。外部指标收集和聚合系统（如 Prometheus）可以访问 HAProxy 路由器指标。您可以在浏览器中以 HTML 的形式和以逗号分隔的值 (CSV) 格式查看 HAProxy 路由器指标。

先决条件

- 您已将防火墙配置为访问默认统计数据端口 1936。

流程

1. 运行以下命令来获取路由器 pod 名称：

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

输出示例

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

2. 获取路由器的用户名和密码，路由器 Pod 存储在 /var/lib/haproxy/conf/metrics-auth/statsUsername 和 /var/lib/haproxy/conf/metrics-auth/statsPassword 文件中：

   a. 运行以下命令来获取用户名：

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

   b. 运行以下命令来获取密码：

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

3. 运行以下命令，获取路由器 IP 和指标证书：

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

4. 运行以下命令，以 Prometheus 格式获取原始统计信息：

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

5.

5. 运行以下命令来安全地访问指标：

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

6. 运行以下命令，访问默认的 stats 端口 1936：

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

例 2.1. 输出示例

```
...
# 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",serv
ice=""} 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. 通过在浏览器中输入以下 URL 来启动 stats 窗口：

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

8. 可选：通过在浏览器中输入以下 URL 来获取 CSV 格式的统计信息：

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

2.2.9.21. 自定义 HAProxy 错误代码响应页面

作为集群管理员，您可以为 503、404 或两个错误页面指定自定义错误代码响应页面。当应用 Pod 没有运行时，HAProxy 路由器会提供一个 503 错误页面，如果请求的 URL 不存在，则 HAProxy 路由器会提供 404 错误页面。例如，如果您自定义 503 错误代码响应页面，则应用 Pod 未运行时会提供页面，并且 HAProxy 路由器为不正确的路由或不存在的路由提供默认的 404 错误代码 HTTP 响应页面。

自定义错误代码响应页面在配置映射中指定，然后修补至 Ingress Controller。配置映射键有两个可用的文件名，如下所示：error-page-503.http 和 error-page-404.http。

自定义 HTTP 错误代码响应页面必须遵循 HAProxy HTTP 错误页面配置指南。以下是默认 OpenShift Dedicated HAProxy 路由器 http 503 错误代码响应页面的示例。您可以使用默认内容作为模板来创建自己的自定义页面。

默认情况下，当应用没有运行或者路由不正确或不存在时，HAProxy 路由器仅提供一个 503 错误页面。这个默认行为与 OpenShift Dedicated 4.8 及更早版本的行为相同。如果没有提供用于自定义 HTTP 错误代码响应的配置映射，且您使用的是自定义 HTTP 错误代码响应页面，路由器会提供默认的 404 或 503 错误代码响应页面。

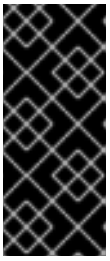
注意

如果您使用 OpenShift Dedicated 默认 503 错误代码页面作为自定义的模板，文件中的标头需要编辑器而不是使用 CRLF 行结尾。

流程

1. 在 openshift-config 命名空间中创建一个名为 my-custom-error-code-pages 的配置映射：

```
$ 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

如果没有为自定义错误代码响应页面指定正确的格式，则会出现路由器 pod 中断。要解决此中断，您必须删除或更正配置映射并删除受影响的路由器 pod，以便使用正确的信息重新创建它们。

2. 对 Ingress Controller 进行补丁以根据名称引用 my-custom-error-code-pages 配置映射：

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

Ingress Operator 将 my-custom-error-code-pages 配置映射从 openshift-config 命名空间复制到 openshift-ingress 命名空间。Operator 根据 openshift-ingress 命名空间中的模式 <your_ingresscontroller_name>-errorpages 命名配置映射。

3. 显示副本：

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

输出示例

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

1 配置映射名称示例为 default-errorpages，因为 default Ingress Controller 自定义资源 (CR) 已被修补。

4. 确认包含自定义错误响应页面的配置映射挂载到路由器卷中，其中配置映射键是具有自定义 HTTP 错误代码响应的文件名：

- 对于 503 自定义 HTTP 自定义错误代码响应：

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

- 对于 404 自定义 HTTP 自定义错误代码响应：

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

验证

验证自定义错误代码 HTTP 响应：

1. 创建测试项目和应用程序：

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

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

2. 对于 503 自定义 http 错误代码响应：

   a. 停止应用的所有容器集。

   b. 运行以下 curl 命令或在浏览器中访问路由主机名：

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

3. 对于 404 自定义 http 错误代码响应：

   a. 访问不存在的路由或路由不正确。

   b. 运行以下 curl 命令或在浏览器中访问路由主机名：

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

4. 检查 haproxy.config 文件中的 errorfile 属性是否正确：

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

2.2.9.22. 设置 Ingress Controller 最大连接数

集群管理员可以设置 OpenShift 路由器部署的最大同时连接数。您可以修补现有的 Ingress Controller 来提高最大连接数。

先决条件

- 以下假设您已创建了 Ingress Controller

流程

- 更新 Ingress Controller，以更改 HAProxy 的最大连接数：

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

警告

如果您设置了大于当前操作系统的 spec.tuningOptions.maxConnections 值，则 HAProxy 进程不会启动。有关这个参数的更多信息，请参阅"Ingress Controller 配置参数"部分中的表。

2.2.10. OpenShift Dedicated Ingress Operator 配置

下表详细介绍了 Ingress Operator 的组件，以及 Red Hat Site Reliability 工程师 (SRE) 在 OpenShift Dedicated 集群上维护此组件。

表 2.3. Ingress Operator 责任图

Ingress 组件	管理方	默认配置？
Scaling Ingress Controller	SRE	是
Ingress Operator thread count	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 3 章 OPENSHIFT DEDICATED 集群的网络验证

当您将 OpenShift Dedicated 集群部署到现有的 Virtual Private Cloud (VPC) 或创建带有集群新子网的额外机器池时，网络验证检查会自动运行。检查会验证您的网络配置并突出显示错误，允许您在部署前解决配置问题。

您还可以手动运行网络验证检查以验证现有集群的配置。

3.1. 了解 OPENSHIFT DEDICATED 集群的网络验证

当您将 OpenShift Dedicated 集群部署到现有的 Virtual Private Cloud (VPC) 中，或使用集群的新子网创建额外的机器池时，网络验证会自动运行。这有助于您在部署前识别并解决配置问题。

当使用 Red Hat OpenShift Cluster Manager 准备安装集群时，会在 Virtual Private Cloud (VPC)子网设置页面中将子网输入到子网 ID 字段中运行自动检查。

当您添加一个机器池，它带有对于集群来说是新的子网时，自动网络验证会检查子网，以确保在置备机器池前可以使用网络连接。

自动网络验证完成后，会将记录发送到服务日志。记录提供验证检查的结果，包括任何网络配置错误。您可以在部署前解决发现的问题，从而使部署成功的机会更大。

您还可以为现有集群手动运行网络验证。这可让您在配置更改后验证集群的网络配置。有关手动运行网络验证检查的步骤，请参阅*手动运行网络验证*。

3.2. 网络验证检查的范围

网络验证包括以下每个要求：

- 存在父虚拟私有云 (VPC)。
- 所有指定子网都属于 VPC。

- VPC 已启用了 enableDnsSupport。
- VPC 已启用了 enableDnsHostnames。
- egress 可用于 AWS 防火墙先决条件 部分中指定的所需域和端口组合。

3.3. 绕过自动网络验证

如果要将带有已知网络配置问题的 OpenShift Dedicated 集群部署到现有的 Virtual Private Cloud (VPC) 中，可以绕过自动网络验证。

如果您在创建集群时绕过网络验证，集群的支持状态是有限的。安装后，您可以解决这个问题，然后手动运行网络验证。在验证成功后，有限的支持状态会被删除。

当使用 Red Hat OpenShift Cluster Manager 将集群安装到现有的 VPC 时，您可以通过在 Virtual Private Cloud (VPC) 子网设置页面中选择 Bypass network verification 来绕过自动验证。

3.4. 手动运行网络验证

您可以使用 Red Hat OpenShift Cluster Manager 手动对现有 OpenShift Dedicated 集群运行网络验证检查。

先决条件

- 您有一个现有的 OpenShift Dedicated 集群。
- 您是集群所有者，或具有集群编辑器角色。

流程

1. 进入到 OpenShift Cluster Manager 并选择您的集群。
2. 从 Actions 下拉菜单中选择 Verify networking。

第 4 章 配置集群范围代理

如果使用现有的 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。

您只能为使用客户云订阅 (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。

4.1. 配置集群范围代理的先决条件

要配置集群范围的代理，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。
- 集群有一个现有的 Virtual Private Cloud (VPC)。
- 您为集群使用客户云订阅 (CCS) 模型。
- 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。

- 您已在 VPC 端点中添加了以下端点：
  - ec2.<aws_region>.amazonaws.com
  - elasticloadbalancing.<aws_region>.amazonaws.com
  - s3.<aws_region>.amazonaws.com

  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而不是在节点级别工作，因此您必须通过 AWS 专用网络将这些请求路由到 AWS EC2 API。在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。

重要

在使用集群范围代理时，您必须将 s3.<aws_region>.amazonaws.com 端点配置为类型 Gateway。

网络要求

- 如果您的代理重新发现出口流量，则必须为域和端口组合创建排除项。下表提供了这些例外的指导。
  - 您的代理必须排除以下 OpenShift URL 的重新加密：

地址	协议/端口	功能
observatorium-mst.api.openshift.com	https/443	必需。用于管理的 OpenShift 特定遥测。
sso.redhat.com	https/443	https://cloud.redhat.com/openshift 站点使用 sso.redhat.com 中的身份验证来下载集群 pull secret，并使用 Red Hat SaaS 解决方案来简化订阅、集群清单和计费报告的监控。

- 您的代理必须排除以下站点可靠性工程 (SRE) 和管理 URL：

地址	协议/端口	功能
*.osdsecuritylogs.splunkcloud.com 或者 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	splunk-forwarder-operator 使用为一个日志转发端点，供 Red Hat SRE 用于基于日志的警报。
http-inputs-osdsecuritylogs.splunkcloud.com	https/443	splunk-forwarder-operator 使用为一个日志转发端点，供 Red Hat SRE 用于基于日志的警报。

其他资源

- 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 上的客户云订阅 或 GCP 上的客户云订阅。

4.2. 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，且未过期

- 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

4.3. 在安装过程中配置代理

当您将带有客户云订阅 (CCS) 集群的 OpenShift Dedicated 安装到现有的 Virtual Private Cloud (VPC) 时，您可以配置 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。

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

如果要将 OpenShift Dedicated 集群安装到现有的 Virtual Private Cloud (VPC) 中，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。您只能为使用客户云订阅 (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步骤，请参阅 *Creating a cluster on AWS with CCS* 或 *Creating a cluster on GCP with CCS*。

其他资源

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

4.5. 安装后配置代理

当您将带有客户云订阅 (CCS) 集群的 OpenShift Dedicated 安装到现有的 Virtual Private Cloud (VPC) 时，您可以配置 HTTP 或 HTTPS 代理。您可以使用 Red Hat OpenShift Cluster Manager 在安装后配置代理。

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

您可以使用 Red Hat OpenShift Cluster Manager 将集群范围的代理配置添加到 Virtual Private Cloud (VPC)的现有 OpenShift Dedicated 集群中。您只能为使用客户云订阅 (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

- 您有一个使用客户云订阅 (CCS) 模型的 OpenShift Dedicated 集群。
- 您的集群部署在 VPC 中。

流程

1. 进入到 OpenShift Cluster Manager 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。如果您使用 TLS-inspecting 代理，则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。无论代理是透明还是需要使用 http-proxy 和 https-proxy 参数显式配置，这个要求都适用。

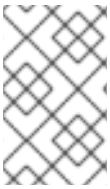
b. 单击 Confirm。

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 5 章 CIDR 范围定义

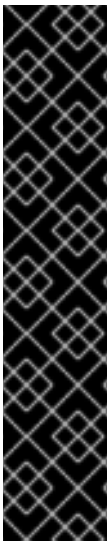
您必须为以下 CIDR 范围指定非重叠范围。



注意

创建集群后无法更改机器 CIDR 范围。

当指定子网 CIDR 范围时，请确保子网 CIDR 范围位于定义的 Machine CIDR 中。您必须根据集群托管的平台，验证子网 CIDR 范围是否允许足够的 IP 地址用于所有预期的工作负载。



重要

OVN-Kubernetes 是 OpenShift Dedicated 4.14 及之后的版本中的默认网络供应商，在内部使用以下 IP 地址范围：100.64.0.0/16、169.254.169.0/29,100.88.0.0/16,fd98::/64,fd69::/125, 和 fd97::/64。如果您的集群使用 OVN-Kubernetes，请不要在集群或基础架构中的任何其他 CIDR 定义中包含这些 IP 地址范围。

对于 OpenShift Dedicated 4.17 及更新的版本，集群使用 169.254.0.0/17 用于 IPv4，fd69::/112 用于 IPv6 作为默认伪装子网。用户也应避免这些范围。对于升级的集群，默认 masquerade 子网没有变化。

5.1. MACHINE CIDR

在 Machine classless inter-domain routing (CIDR)字段中，您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。子网必须是连续的。单个可用区部署支持最少有 128 个地址的 IP 地址范围（使用子网前缀 /25）。多可用区部署支持最少 256 个地址的 IP 地址范围（使用子网前缀 /24）。

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。

5.2. SERVICE CIDR

在 Service CIDR 字段中，您必须为服务指定 IP 地址范围。建议（但不是必需的）地址块在集群之间是相同的。这将不会创建 IP 地址冲突。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。

5.3. POD CIDR

在 pod CIDR 字段中，您必须为 pod 指定 IP 地址范围。

建议（但不是必需的）地址块在集群之间是相同的。这将不会创建 IP 地址冲突。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。

5.4. 主机前缀

在 Host Prefix 字段中，您必须指定分配给调度到各个机器的 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 地址范围中分配一个 /23 子网。默认值为 /23，允许 512 个集群节点以及每个节点的 512 个 pod （其中两个超过我们的最大支持）。

第 6 章 网络安全性

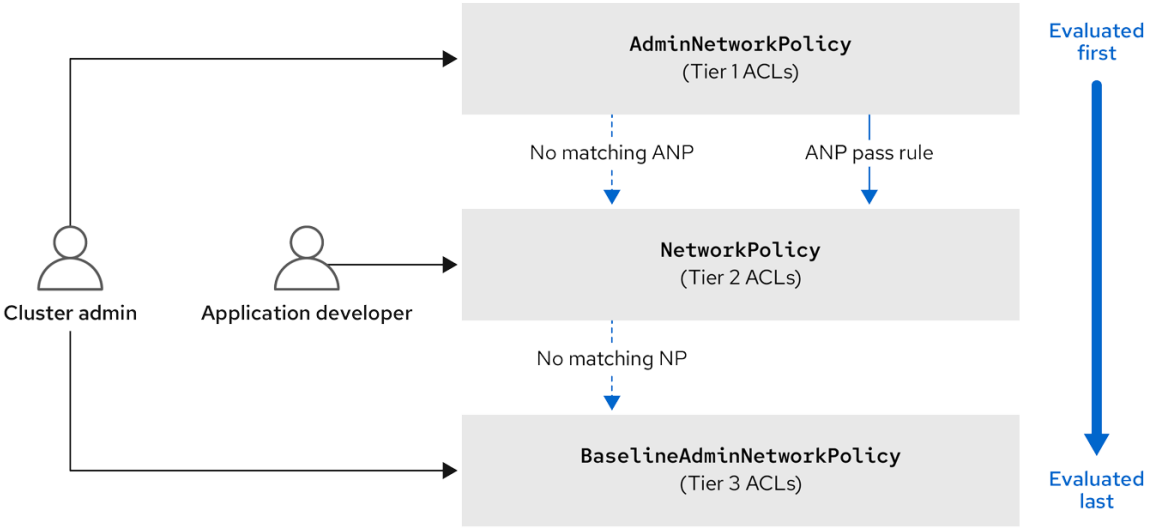
6.1. 了解网络策略 API

Kubernetes 提供了两个用户可用于强制实施网络安全的功能。允许用户强制执行网络策略的一个功能是 NetworkPolicy API，主要用于应用程序开发人员和命名空间租户，通过创建命名空间范围的策略来保护其命名空间。

第二个功能是 AdminNetworkPolicy，它由两个 API 组成：AdminNetworkPolicy (ANP) API 和 BaselineAdminNetworkPolicy (BANP) API。ANP 和 BANP 是为集群和网络管理员设计的，以通过创建集群范围的策略来保护其整个集群。集群管理员可以使用 ANPs 来强制实施优先于 NetworkPolicy 对象的不可覆盖的策略。管理员可以使用 BANP 设置并强制实施可选的集群范围的网络策略规则，当需要时，用户可以使用 NetworkPolicy 对象覆盖它。当一起使用时，ANP、BANP 和网络策略可以实现完整的多租户隔离，管理员可用于保护其集群。

OpenShift Dedicated 中的 OVN-Kubernetes CNI 使用访问控制列表(ACL)层来评估和应用这些网络策略。ACL 按照从 Tier 1 到 Tier 3 的降序进行评估。

第 1 级评估 AdminNetworkPolicy (ANP)对象。第 2 级评估 NetworkPolicy 对象。第 3 级评估 BaselineAdminNetworkPolicy (BANP)对象。



首先评估 ANP。当匹配是 ANP allow 或 deny 规则时，集群中的任何现有 NetworkPolicy 和 BaselineAdminNetworkPolicy（BANP）对象将不会被评估。当匹配是 ANP pass，评估会从 ACL 的第 1 层移到第 2 层，在其中评估 NetworkPolicy 策略。如果没有 NetworkPolicy 与流量匹配，则评估从第 2 层 ACL 移到评估 BANP 的第 3 层 ACL。

6.1.1. AdminNetworkPolicy 和 NetworkPolicy 自定义资源之间的主要区别

下表解释了集群范围的 AdminNetworkPolicy API 和命名空间范围 NetworkPolicy API 之间的主要区别。

策略元素	AdminNetworkPolicy	NetworkPolicy
适用的用户	集群管理员或等同功能	命名空间所有者
影响范围	Cluster	namespaced
丢弃流量	支持将显式 **Deny** 操作设置为规则。	在策略创建时通过隐式 **Deny** 隔离支持。
委派流量	支持 **Pass** 操作集作为一个规则。	Not applicable
允许流量	支持将显式 **Allow** action 设置为规则。	所有规则的默认操作都是 allow。
策略中的规则优先级	取决于它们出现在 ANP 中的顺序。规则在优先级越高的位置越高。	规则是添加的
策略优先级	在 ANPs 中，**priority** 字段设置评估的顺序。策略优先级越低的优先级优先级越低。	策略之间没有策略排序。
功能优先级	首先通过 1 层 ACL 和 BANP 评估，最后通过第 3 层 ACL 评估。	在 ANP 和 BANP 之前强制实施，它们会在 ACL 层 2 中进行评估。
匹配 pod 选择	可以在命名空间之间应用不同的规则。	可以在单一命名空间中的 pod 之间应用不同的规则。
集群出口流量	通过 **节点 和网络** 对等点支持	通过 **ipBlock** 字段以及接受的 CIDR 语法支持。
集群入口流量	不支持	不支持
完全限定域名(FQDN)对等支持	不支持	不支持
命名空间选择器	支持通过使用 **namespaces.matchLabels** 字段进行命名空间的高级选择	支持使用 **namespaceSelector** 字段支持基于标签的命名空间选择

6.2. 网络策略

6.2.1. 关于网络策略

作为开发者，您可以定义网络策略来限制集群中 pod 的流量。

6.2.1.1. 关于网络策略

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。

网络策略仅适用于 TCP、UDP、ICMP 和 SCTP 协议。其他协议不会受到影响。

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。但是，连接到 host-networked pod 的 pod 会受到网络策略规则的影响。

网络策略无法阻止来自 localhost 或来自其驻留的节点的流量。

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：

重要

要允许同一命名空间中的 hostNetwork pod 的入站连接，您需要将 allow-from-hostnetwork 策略与 allow-same-namespace 策略一起应用。

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。

6.2.1.1.1. 使用 allow-from-router 网络策略

使用以下 NetworkPolicy 来允许外部流量，而不考虑路由器配置：

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

1

policy-group.network.openshift.io/ingress:"" 标签支持 OVN-Kubernetes。

6.2.1.1.2. 使用 allow-from-hostnetwork 网络策略

添加以下 allow-from-hostnetwork NetworkPolicy 对象来定向来自主机网络 pod 的流量。

```
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
```

6.2.1.2. 使用 OVN-Kubernetes 网络插件优化网络策略

在设计您的网络策略时，请参考以下指南：

- 对于具有相同 spec.podSelector spec 的网络策略，使用带有多个 ingress 或 egress 规则的一个网络策略比带有 ingress 或 egress 子集的多个网络策略更高效。

- 每个基于 podSelector 或 namespaceSelector spec 的 ingress 或 egress 规则会生成一个的 OVS 流数量，它与由网络策略选择的 pod 数量 + 由 ingress 或 egress 选择的 pod 数量成比例因此，最好使用在一个规则中可以选择您所需的 pod 的 podSelector 或 namespaceSelector 规格，而不是为每个 pod 创建单独的规则。

  例如，以下策略包含两个规则：

```
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
```

  以下策略表示这两个规则与以下相同的规则：

```
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]}
```

  相同的指南信息适用于 spec.podSelector spec。如果不同的网络策略有相同的 ingress 或 egress 规则，则创建一个带有通用的 spec.podSelector spec 可能更有效率。例如，以下两个策略有不同的规则：

```
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
```

以下网络策略将这两个相同的规则作为一个：

```
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
```

当只有多个选择器表示为一个选择器时，您可以应用此优化。如果选择器基于不同的标签，则可能无法应用此优化。在这些情况下，请考虑为网络策略优化应用一些新标签。

6.2.1.3. 后续步骤

- 创建网络策略

6.2.2. 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。

6.2.2.1. 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。

6.2.2.2. 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 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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

这是一个基本的策略，阻止配置其他网络策略所允许的跨 pod 流量以外的所有跨 pod 网络。

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

允许从特定命名空间中到一个 pod 的入口流量

此策略允许流量从在 namespace-y 中运行的容器集到标记 pod-a 的 pod。

```
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. 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/deny-by-default 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。

6.2.2.3. 创建默认拒绝所有网络策略

这是一个基本的策略，阻止其他部署网络策略允许的网络流量以外的所有跨 pod 网络。此流程强制使用默认 deny-by-default 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 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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. 创建以下 YAML，以定义 deny-by-default 策略，以拒绝所有命名空间中的所有 pod 的入口流量。将 YAML 保存到 deny-by-default.yaml 文件中：

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

1 namespace：default 将此策略部署到 default 命名空间。

2 podSelector: 为空，这意味着它与所有 pod 匹配。因此，该策略适用于 default 命名空间中的所有 pod。

3

没有指定 ingress 规则。这会导致传入的流量丢弃至所有 pod。

2. 输入以下命令应用策略：

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

输出示例

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

6.2.2.4. 创建网络策略以允许来自外部客户端的流量

使用 deny-by-default 策略，您可以继续配置策略，允许从外部客户端到带有标签 app=web 的 pod 的流量。

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。

按照以下步骤配置策略，以直接从公共互联网允许外部服务，或使用 Load Balancer 访问 pod。只有具有标签 app=web 的 pod 才允许流量。

先决条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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，以直接从公共互联网的流量或使用负载均衡器访问 pod。将 YAML 保存到 web-allow-external.yaml 文件中：

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

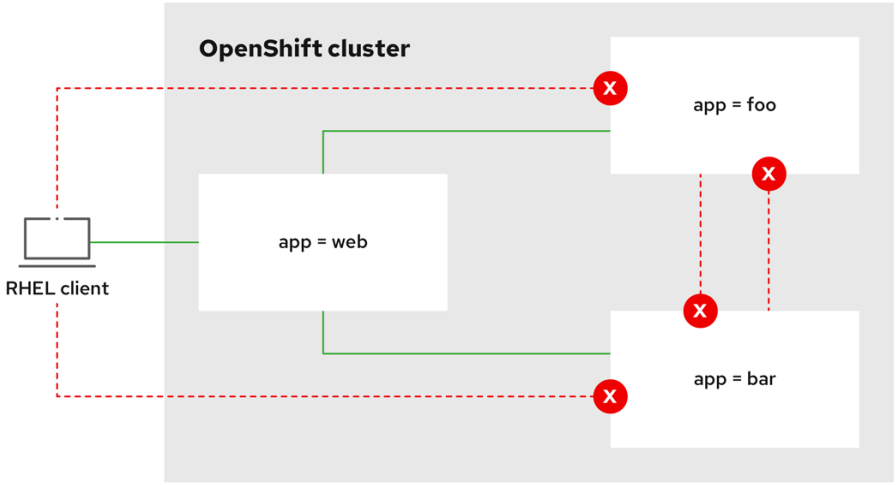
2. 输入以下命令应用策略：

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

输出示例

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

此策略允许来自所有资源的流量，包括下图所示的外部流量：



292_OpenShift_1122

6.2.2.5. 创建网络策略，允许从所有命名空间中到应用程序的流量

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。

按照以下步骤配置允许从所有命名空间中的所有 pod 流量到特定应用程序的策略。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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

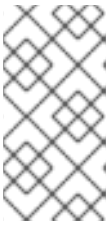
1. 创建一个策略，允许从所有命名空间中的所有 pod 流量到特定应用。将 YAML 保存到 web-allow-all-namespaces.yaml 文件中：

```
-
```

```
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 仅将策略应用到 default 命名空间中的 app:web pod。

2 选择所有命名空间中的所有 pod。



注意

默认情况下，如果您省略了指定 namespaceSelector 而不是选择任何命名空间，这意味着策略只允许从网络策略部署到的命名空间的流量。

2. 输入以下命令应用策略：

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

输出示例

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

验证

1

1. 输入以下命令在 default 命名空间中启动 web 服务：

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

2. 运行以下命令在 secondary 命名空间中部署 alpine 镜像并启动 shell：

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

3. 在 shell 中运行以下命令，并观察是否允许请求：

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

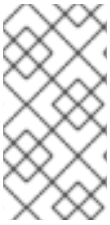
预期输出

```
<!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>
```

6.2.2.6. 创建网络策略，允许从一个命名空间中到应用程序的流量



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。

按照以下步骤配置允许从特定命名空间中到带有 app=web 标签的 pod 的策略。您可能需要进行以下操作：

- 将流量限制为部署生产工作负载的命名空间。
- 启用部署到特定命名空间的监控工具，以从当前命名空间中提取指标。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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. 创建一个策略，允许来自特定命名空间中所有 pod 的流量，其标签为 purpose=production。将 YAML 保存到 web-allow-prod.yaml 文件中：

```
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 仅将策略应用到 default 命名空间中的 app:web pod。

2 将流量仅限制为具有标签 purpose=production 的命名空间中的 pod。

2. 输入以下命令应用策略：

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

输出示例

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

验证

1. 输入以下命令在 default 命名空间中启动 web 服务：

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

2. 运行以下命令来创建 prod 命名空间：

```
$ oc create namespace prod
```

3. 运行以下命令来标记 prod 命名空间：

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

4. 运行以下命令来创建 dev 命名空间：

```
$ oc create namespace dev
```

5. 运行以下命令来标记 dev 命名空间：

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

6. 运行以下命令在 dev 命名空间中部署 alpine 镜像并启动 shell：

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

7. 在 shell 中运行以下命令，并观察请求是否被阻止：

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

预期输出

```
wget: download timed out
```

8. 运行以下命令，在 prod 命名空间中部署 alpine 镜像并启动 shell：

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

9. 在 shell 中运行以下命令，并观察是否允许请求：

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

预期输出

```
<!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>
```

6.2.2.7. 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

- 已登陆到 OpenShift Cluster Manager。
- 您创建了 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 集群中创建一个项目。

流程

1. 在 OpenShift Cluster Manager 中点您要访问的集群。
2. 点 Open console 以进入到 OpenShift Web 控制台。
3. 点身份提供程序，并提供您的凭证以登录到集群。
4. 使用管理员视角，在 Networking 下点 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： 您可以通过选择 Deny all ingress traffic 或 Deny all egress traffic 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。此操作在 Add allowed source 下拉菜单中创建一个新的 Ingress rule 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：
       - Allow pods from the same namespace 将流量限制为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。

- Allow pods from inside the cluster 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。

- Allow peers by IP block 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：

- Allow pods from the same namespace 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。

- Allow pods from inside the cluster 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。

- Allow peers by IP block 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。

6.2.3. 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。

6.2.3.1. 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。

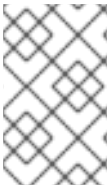
2 描述策略应用到的 pod 的选择器。策略对象只能选择定义 NetworkPolicy 对象的项目中的 pod。

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。

6.2.3.2. 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。

前提条件

- 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>

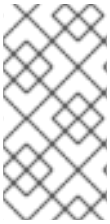
可选： 如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

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。

6.2.3.3. 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。

前提条件

- 已登陆到 OpenShift Cluster Manager。
- 您创建了 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 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

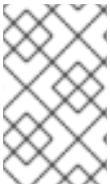
您只能查看这些策略的详情。您不能编辑这些策略。

6.2.4. 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。

6.2.4.1. 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。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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。

6.2.4.2. 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。

前提条件

- 已登陆到 OpenShift Cluster Manager。
- 您创建了 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 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 Delete NetworkPolicy。
   - 使用独立网络策略详情中的 Actions 下拉菜单删除策略：

a. 点网络策略的 Actions 下拉菜单。

b. 从菜单中选择 Delete NetworkPolicy。

6.2.5. 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

如本节所述配置网络策略，提供了与 OpenShift Dedicated 的早期版本中 OpenShift SDN 的多租户模式类似的网络隔离。

6.2.5.1. 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。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件，并设置 mode: NetworkPolicy。
- 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 OVN-Kubernetes 的首选命名空间选择器标签。

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 新的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 7 章 OVN-KUBERNETES 网络插件

7.1. 关于 OVN-KUBERNETES 网络插件

OpenShift Dedicated 集群将虚拟网络用于 pod 和服务网络。

Red Hat OpenShift Networking 的一部分，OVN-Kubernetes 网络插件是 OpenShift Dedicated 的默认网络供应商。OVN-Kubernetes 基于 Open Virtual Network（OVN），它提供了一个基于 overlay 的网络实现。使用 OVN-Kubernetes 插件的集群还在每个节点上运行 Open vSwitch (OVS)。OVN 在每个节点上配置 OVS 来实现声明的网络配置。

注意

OVN-Kubernetes 是 OpenShift Dedicated 和单节点 OpenShift 部署的默认网络解决方案。

OVN-Kubernetes （来自 OVS 项目）使用许多相同的结构，如开放流规则，以确定数据包通过网络传输的方式。如需更多信息，请参阅 Open Virtual Network 网站。

OVN-Kubernetes 是 OVS 的一系列守护进程，用于将虚拟网络配置转换为 OpenFlow 规则。OpenFlow 是一种用于与网络交换机和路由器通信的协议，为远程控制网络设备上的网络流量流提供途径，以便网络管理员能够配置、管理和监控网络流量的流。

OVN-Kubernetes 提供了 OpenFlow 提供的更多高级功能。OVN 支持分布式虚拟路由、分布式逻辑交换机、访问控制、动态主机配置协议(DHCP)和 DNS。OVN 在逻辑流中实施分布式虚拟路由，它们等同于开放流。例如，如果您有一个 pod 将 DHCP 请求发送到网络上的 DHCP 服务器，则请求中的逻辑流规则可帮助 OVN-Kubernetes 处理数据包，以便服务器可以响应网关、DNS 服务器、IP 地址和其他信息。

OVN-Kubernetes 在每个节点上运行一个守护进程。数据库和 OVN 控制器都有守护进程集，每个节点上运行的 OVN 控制器。OVN 控制器在节点上对 Open vSwitch 守护进程进行编程，以支持网络提供程序功能：出口 IP、防火墙、路由器、混合网络、IPSEC 加密、IPv6 加密、网络策略日志、网络策略日志、硬件卸载和多播。

7.1.1. OVN-Kubernetes 目的

OVN-Kubernetes 网络插件是一个开源、功能齐全的 Kubernetes CNI 插件，它使用 Open Virtual Network (OVN)来管理网络流量。OVN 是一个社区开发、与供应商无关的网络虚拟化解决方案。OVN-Kubernetes 网络插件使用以下技术：

- OVN 管理网络流量流。
- Kubernetes 网络策略支持和日志，包括入口和出口规则。
- 通用网络虚拟化封装(Geneve)协议，而不是虚拟可扩展局域网(VXLAN)，以在节点之间创建覆盖网络。

OVN-Kubernetes 网络插件支持以下功能：

- 可以运行 Linux 和 Microsoft Windows 工作负载的混合集群。此环境称为*混合网络*。
- 将网络数据处理从主机中央处理单元(CPU)卸载到兼容的网卡和数据处理单元(DPU)。这称为*硬件卸载（hardware offloading）*。
- IPv4-primary 双栈网络，在裸机、VMware vSphere、IBM Power®、IBM Z® 和 RHOSP 平台上。
- 裸机平台上的 IPv6 单堆栈网络。
- 在裸机、VMware vSphere 或 RHOSP 平台上运行的集群的 IPv6-primary 双栈网络。
- 出口防火墙设备和出口 IP 地址。
- 以重定向模式运行的出口路由器设备。
- 集群内通信的 IPsec 加密。

7.1.2. OVN-Kubernetes IPv6 和双栈限制

OVN-Kubernetes 网络插件有以下限制：

- 对于为双栈网络配置的集群，IPv4 和 IPv6 流量都必须使用与默认网关相同的网络接口。如果不满足此要求，则 ovnkube-node 守护进程集中的主机上的容器集进入 CrashLoopBackOff 状态。如果您使用 oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml 等命令显示 pod，则 status 字段包含多个有关默认网关的消息，如以下输出所示：

```
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
```

唯一的解析是重新配置主机网络，以便两个 IP 系列都针对默认网关使用相同的网络接口。

- 对于为双栈网络配置的集群，IPv4 和 IPv6 路由表必须包含默认网关。如果不满足此要求，则 ovnkube-node 守护进程集中的主机上的容器集进入 CrashLoopBackOff 状态。如果您使用 oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml 等命令显示 pod，则 status 字段包含多个有关默认网关的消息，如以下输出所示：

```
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
```

唯一的解析是重新配置主机网络，以便两个 IP 系列都包含默认网关。

7.1.3. 会话关联性

会话关联性是适用于 Kubernetes Service 对象的功能。如果要确保每次连接到 <service_VIP>:<Port> 时，您可以使用 *会话关联性*，流量始终被加载到同一后端。如需更多信息，包括如何根据客户端的 IP 地址设置会话关联性，请参阅会话关联性。

会话关联性的粘性超时

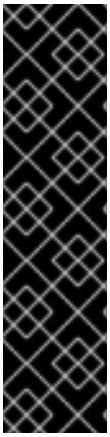
OpenShift Dedicated 的 OVN-Kubernetes 网络插件根据最后一个数据包计算来自客户端的会话的粘性超时。例如，如果您运行 curl 命令 10 次，则粘性会话计时器从第十个数据包开始，而不是第一个数据包。因此，如果客户端不断联系该服务，则会话永远不会超时。当服务没有收到 timeoutSeconds 参数所设定的时间的数据包时，超时开始。

第 8 章 OPENSHIFT SDN 网络插件

8.1. 为项目启用多播

8.1.1. 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，网络策略会影响命名空间中的所有连接。但是，多播不受网络策略的影响。如果在与网络策略相同的命名空间中启用了多播，则始终允许多播，即使有一个 deny-all 网络策略。在启用网络策略前，集群管理员应考虑对多播的影响。

默认情况下，OpenShift Dedicated pod 间的多播流量被禁用。如果使用 OVN-Kubernetes 网络插件，可以根据每个项目启用多播。

8.1.2. 启用 pod 间多播

您可以为项目启用 pod 间多播。

前提条件

- 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 namespace <namespace> \
  k8s.ovn.org/multicast-enabled=true
```

提示

您还可以应用以下 YAML 来添加注解：

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/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/ubi9
      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/ubi9
      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 9 章 配置路由

9.1. 路由配置

9.1.1. 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。

前提条件

- 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
```

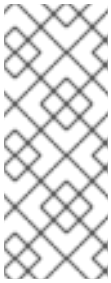
1 在本例中，路由名为 hello-openshift。

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 对象是在安装过程中创建的，且无法更改。如果要指定不同的域，您可以使用 appsDomain 选项指定备选集群域。

2 targetPort 是由此路由指向的服务选择的 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}
```

9.1.2. 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。

前提条件

- 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
```

9.1.3. 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。

9.1.3.1. 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。

前提条件

- 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 OpenShift CLI（oc）。

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 <route_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，大约为 8.5 小时。

注意

在这个示例中，等号 (=) 包括在引号里。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。

9.1.3.2. 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。

前提条件

- 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 OpenShift CLI（oc）。

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 <route_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
```

9.1.4. 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 控制器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 告知入口控制器处理会话，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。

9.1.4.1. 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。删除 Cookie 可强制下一请求重新选择端点。结果是，如果服务器过载，该服务器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
```

9.1.5. 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。

下表显示了路由及其可访问性示例：

表 9.1. 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。

9.1.6. HTTP 标头配置

OpenShift Dedicated 提供了不同的使用 HTTP 标头的方法。在设置或删除标头时，您可以使用 Ingress Controller 中的特定字段或单独的路由来修改请求和响应标头。您还可以使用路由注解设置某些标头。配置标头的各种方法在协同工作时可能会带来挑战。

注意

您只能在 IngressController 或 Route CR 中设置或删除标头，您无法附加它们。如果使用值设置 HTTP 标头，则该值必须已完成，且在以后不需要附加。在附加标头（如 X-Forwarded-For 标头）时，请使用 spec.httpHeaders.forwardedHeaderPolicy 字段，而不是 spec.httpHeaders.actions。

9.1.6.1. 优先级顺序

当在 Ingress Controller 和路由中修改相同的 HTTP 标头时，HAProxy 会根据它是请求还是响应标头来优先选择操作。

- 对于 HTTP 响应标头，Ingress Controller 中指定的操作会在路由中指定的操作后执行。这意味着 Ingress Controller 中指定的操作具有优先权。

- 对于 HTTP 请求标头，路由中指定的操作会在 Ingress Controller 中指定的操作后执行。这意味着路由中指定的操作具有优先权。

例如，集群管理员使用以下配置设置 X-Frame-Options 响应标头，其值为 DENY ：

IngressController spec 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
```

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

路由所有者设置 Ingress Controller 中设置的相同响应标头，但使用以下配置值 SAMEORIGIN ：

Route 规格示例

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

当 IngressController spec 和 Route spec 都配置 X-Frame-Options 响应标头时，Ingress Controller 的全局级别上为此标头设置的值具有优先权，即使一个特定的路由允许帧。对于请求标头，Route spec 值会覆盖 IngressController spec 值。

这是因为 haproxy.config 文件使用以下逻辑，其中 Ingress Controller 被视为前端，单个路由被视为后端。应用到前端配置的标头值 DENY 使用后端中设置的值 SAMEORIGIN 覆盖相同的标头：

```
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'
```

另外，Ingress Controller 或路由中定义的任何操作都覆盖使用路由注解设置的值。

9.1.6.2. 特殊情况标头

以下标头可能会阻止完全被设置或删除，或者在特定情况下允许：

表 9.2. 特殊情况标头配置选项

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
proxy	否	否	**proxy** HTTP 请求标头可以通过将标头值注入 **HTTP_PROXY** 环境变量来利用这个安全漏洞的 CGI 应用程序。**proxy** HTTP 请求标头也是非标准的，在配置期间容易出错。	否
主机	否	是	当使用 **IngressController** CR 设置 **host** HTTP 请求标头时，HAProxy 在查找正确的路由时可能会失败。	否
strict-transport-security	否	否	**strict-transport-security** HTTP 响应标头已使用路由注解处理，不需要单独的实现。	是：**haproxy.router.openshift.io/hsts_header** 路由注解

标头名称	使用 IngressController spec 进行配置	使用 Route 规格进行配置	禁止的原因	使用其他方法进行配置
cookie 和 **set-cookie**	否	否	HAProxy 集的 Cookie 用于会话跟踪，用于将客户端连接映射到特定的后端服务器。允许设置这些标头可能会影响 HAProxy 的会话关联，并限制 HAProxy 的 Cookie 的所有权。	是： • **haproxy.router.openshift.io/disable_cookie** 路由注解 • **haproxy.router.openshift.io/cookie_name** 路由注解

9.1.7. 在路由中设置或删除 HTTP 请求和响应标头

出于合规的原因，您可以设置或删除某些 HTTP 请求和响应标头。您可以为 Ingress Controller 提供的所有路由或特定路由设置或删除这些标头。

例如，如果内容使用多种语言编写，您可能希望让 Web 应用程序在备用位置提供内容，即使 Ingress Controller 为路由指定的默认全局位置。

以下流程会创建一个设置 Content-Location HTTP 请求标头的路由，以便与应用程序关联的 URL https://app.example.com 定向到位置 https://app.example.com/lang/en-us。将应用程序流量定向到此位置意味着使用该特定路由的任何人都可以访问以美国英语编写的 Web 内容。

先决条件

- 已安装 OpenShift CLI(oc)。
- 以项目管理员身份登录到 OpenShift Dedicated 集群。
- 您有一个 web 应用来公开端口，以及侦听端口流量的 HTTP 或 TLS 端点。

流程

1. 创建一个路由定义，并将它保存到名为 app-example-route.yaml 的文件中：

使用 HTTP 标头指令创建路由的 YAML 定义

```yaml
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 要在 HTTP 标头上执行的操作列表。

2 您要更改的标头类型。在本例中，响应标头。

3 您要更改的标头的名称。有关您可以设置或删除的可用标头列表，请参阅 *HTTP 标头配置*。

4 在标头中执行的操作类型。此字段可以具有 Set 或 Delete 的值。

5

在设置 HTTP 标头时，您必须提供一个 value。该值可以是该标头的可用指令列表中的字符串，如 DENY，也可以是使用 HAProxy 的动态值语法来解释的动态值。在这种情况下，该值被设置为内容的相对位置。

2. 使用新创建的路由定义，创建到现有 Web 应用程序的路由：

```
$ oc -n app-example create -f app-example-route.yaml
```

对于 HTTP 请求标头，路由定义中指定的操作会在 Ingress Controller 中对 HTTP 请求标头执行的任何操作后执行。这意味着，路由中这些请求标头设置的任何值都将优先于 Ingress Controller 中设置的值。有关 HTTP 标头处理顺序的更多信息，请参阅 *HTTP 标头配置*。

9.1.8. 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 9.3. 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。对于 TLS 透传路由，默认值为 source。对于所有其他路由，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。	

变量	描述	默认的环境变量
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。	
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： 使用此注解提供了对拒绝服务攻击的基本保护。	
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： 使用此注解提供了对拒绝服务攻击的基本保护。	
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： 使用此注解提供了对拒绝服务攻击的基本保护。	
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： 使用此注解提供了对拒绝服务攻击的基本保护。	
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**

变量	描述	默认的环境变量
router.openshift.io/haproxy.health.check.interval	为后端健康检查设定间隔。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	为路由设置允许列表。允许列表（allowlist）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。不是来自允许列表中的 IP 地址的请求会被丢弃。 在 **haproxy.config** 文件中直接可见的最大 IP 地址和 CIDR 范围数为 61. [1]	
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/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**: 浏览器不会在跨站点请求上发送 Cookie，当用户从外部站点导航到原始站点时发送 Cookie。当未指定 **SameSite** 值时，这是默认的浏览器行为。 **Strict** ：浏览器仅针对同一站点请求发送 Cookie。 **None** ：浏览器为跨站点和相同站点请求发送 Cookie。 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**

1. **如果允许列表中的 IP 地址和 CIDR 范围超过 61，它们将被写入到一个独立的文件中，haproxy.config 会引用这个文件。此文件存储在 var/lib/haproxy/router/whitelists 文件夹中。**

注意

为确保地址被写入允许列表，请检查 Ingress Controller 配置文件中是否列出了 CIDR 范围的完整列表。etcd 对象大小限制了路由注解的大小。因此，它实际上是为您可以在允许列表中包含的 IP 地址和 CIDR 范围的最大数量创建一个阈值。

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。
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 9.4. 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/
/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

haproxy.router.openshift.io/rewrite-target 中的某些特殊字符需要特殊处理，因为它们必须正确转义。请参阅下表以了解这些字符的处理方式。

表 9.5. 特殊字符处理

对于字符	使用字符	注
#	\#	避免使用 #，因为它会终止重写表达式
%	% 或 %%	避免奇数序列，如 %%%
'	\'	避免 '，因为它被忽略

所有其他有效的 URL 字符可以在不转义的情况下使用。

9.1.9. 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 会生成一个不安全的路由。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：

前提条件

- 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 使用此精确的语法指定 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，后跟一个随机的后缀。

2 要使用默认证书，路由不应指定 spec.certificate。

3 路由应指定 edge 终止策略。

9.1.10. 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。

前提条件

- 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-----
...
```

其他资源

- 使用 appsDomain 选项指定备选集群域

9.2. 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。

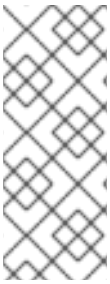
9.2.1. 使用自定义证书创建重新加密路由

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。

前提条件

- 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-----
```

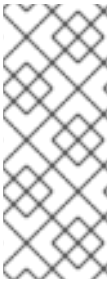
如需了解更多选项，请参阅 oc create route reencrypt --help。

9.2.2. 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。

前提条件

- 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。

9.2.3. 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。

前提条件

- 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)

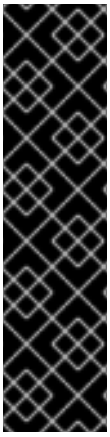
termination 字段设置为 passthrough。这是唯一需要 tls 的字段。

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。

9.2.4. 使用外部受管证书创建路由

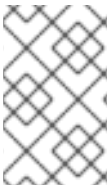


重要

在 TLS secret 中使用外部证书保护路由只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以使用路由 API 的 .spec.tls.externalCertificate 字段来使用第三方证书管理解决方案配置 OpenShift Dedicated 路由。您可以通过 secret 引用外部管理的 TLS 证书，无需手动证书管理。使用外部受管证书可减少确保证书更新平稳推出的错误，使 OpenShift 路由器能够及时提供更新的证书。



注意

此功能适用于边缘路由和重新加密路由。

前提条件

- 您必须启用 RouteExternalCertificate 功能门。
- 您必须在 routes/custom-host 上具有 create 和 update 权限。
- 您必须有一个包含 PEM 编码格式的有效证书/密钥对的 secret，类型为 kubernetes.io/tls，

其中包括 tls.key 和 tls.crt 键。

- 您必须将引用的 secret 放在与您要保护的路由相同的命名空间中。

流程

1. 运行以下命令，在与 secret 相同的命名空间中创建 角色，以允许路由器服务帐户读取访问权限：

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-
name=<secret-name> \ 1
--namespace=<current-namespace> 2
```

1. 指定 secret 的实际名称。

2. 指定 secret 和路由所在的命名空间。

2. 运行以下命令，在与 secret 相同的命名空间中创建 rolebinding，并将 router 服务帐户绑定到新创建的角色：

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace> 1
```

1. 指定 secret 和路由所在的命名空间。

3. 创建一个定义 路由的 YAML 文件，并使用以下示例指定包含证书的 secret。

安全路由的 YAML 定义

```
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 指定 secret 的实际名称。

4. 运行以下命令来创建路由资源：

```
$ oc apply -f <route.yaml> 1
```

1 指定生成的 YAML 文件名。

如果 secret 存在并具有证书/密钥对，如果满足所有先决条件，路由器将提供生成的证书。

注意

如果没有提供 .spec.tls.externalCertificate，路由器将使用默认生成的证书。

使用 .spec.tls.externalCertificate 字段时，您无法提供 .spec.tls.certificate 字段或 .spec.tls.tls.key 字段。

其他资源

- 有关使用外部管理证书对路由进行故障排除，请检查 OpenShift Dedicated 路由器 pod 日志中的错误，请参阅 调查 pod 问题。