Red Hat OpenShift Service on AWS 4

网络

配置 Red Hat OpenShift Service on AWS 网络

Last Updated: 2024-07-02

Red Hat OpenShift Service on AWS 4 网络

配置 Red Hat OpenShift Service on AWS 网络

法律通告

摘要

本文档提供有关 Red Hat OpenShift Service on AWS（ROSA） 集群的安全信息。

目录

第 1 章 RED HAT OPENSHIFT SERVICE ON AWS 中的 DNS OPERATOR

在 Red Hat OpenShift Service on AWS 中，DNS Operator 会部署和管理 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。

1.1. 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**)。如果转发区是由 Red Hat OpenShift Service on AWS 管理的入口域，则必须为域授权上游名称服务器。

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。

第 2 章 RED HAT OPENSHIFT SERVICE ON AWS 中的 INGRESS OPERATOR

2.1. RED HAT OPENSHIFT SERVICE ON AWS 中的 INGRESS OPERATOR

在创建 Red Hat OpenShift Service on AWS 集群时，在集群中运行的 Pod 和服务会各自分配自己的 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 实现 **IngressController** API，是负责启用对 Red Hat OpenShift Service on AWS 集群服务的外部访问的组件。

Ingress Operator 通过部署和管理一个或多个基于 HAProxy 的 Ingress Controller 来处理路由，使外部客户端可以访问您的服务。Red Hat Site Reliability Engineers (SRE) 为 Red Hat OpenShift Service on AWS 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. 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。

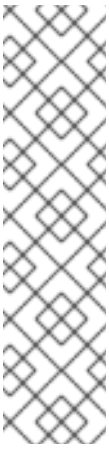
2.3. INGRESS CONTROLLER 配置参数

ingresscontrollers.operator.openshift.io 资源提供了以下配置参数。

参数	描述

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 控制器副本数量。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。 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)  **注意** **HostNetwork** 有一个 **hostNetwork** 字段，它有以下用于可选绑定端口的默认值：**httpPort: 80**,**httpsPort: 443**, 和 **statsPort: 1936**。使用绑定端口，您可以为 **HostNetwork** 策略在同一节点上部署多个 Ingress Controller。 **Example** apiVersion: operator.openshift.io/v1 kind: IngressController metadata: name: internal namespace: openshift-ingress-operator spec: domain: example.com endpointPublishingStrategy: type: HostNetwork hostNetwork: httpPort: 80 httpsPort: 443 statsPort: 1936  **注意** 在 Red Hat OpenStack Platform (RHOSP) 上，只有云供应商配置为创建运行状况监视器时，才会支持 **LoadBalancerService** 端点发布策略。对于 RHOSP 16.2，只有在您使用 Amphora Octavia 供应商时，才能使用此策略。 如需更多信息，请参阅 RHOSP 安装文档中的"设置云供应商选项"部分。

参数	描述
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。 in-use 证书（无论是生成的证书还是用户指定的证书）会自动与 AWS 内置的 OAuth 服务器上的 Red Hat OpenShift Service 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。
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 ```

参数	描述
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**。
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。

参数	描述
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。

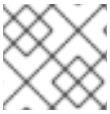
参数	描述
IngressControllerLogging	**logging** 定义了有关在哪里记录什么内容的参数。如果此字段为空，则会启用运行日志，但禁用访问日志。 • **access** 描述了客户端请求的日志记录方式。如果此字段为空，则禁用访问日志。 ◦ **destination** 描述日志消息的目的地。 ▪ **type** 是日志的目的地类型： • **Container** 指定日志应该进入 sidecar 容器。Ingress Operator 在 Ingress Controller pod 上配置名为 **logs** 的容器，并配置 Ingress Controller 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

参数	描述
	Ingress Controller 连接会话保留多少内存（以字节为单位），用于 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。



注意

所有参数都是可选的。

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。

2.3.1.1. 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 Red Hat OpenShift Service on AWS 组件需要哪些 TLS 密码。Red Hat OpenShift Service on AWS 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。

2.3.1.2. 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
...
```

2.3.1.3. 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
```

2.4. 查看默认的 INGRESS CONTROLLER

Ingress Operator 是 Red Hat OpenShift Service on AWS 的一个核心功能，开箱即用。

每个 Red Hat OpenShift Service on AWS 新安装都有一个名为 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
```

2.5. 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
```

2.6. 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>
```

2.7. 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>
```

2.8. 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 Red Hat OpenShift Service on AWS 更新过程中可能会改变，因此当维护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。要进一步定制您的自定义 Ingress Controller，请参阅"配置 Ingress Controller"。

前提条件

- 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. 为 **IngressController** 对象指定自定义**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（在域中添加了 ***.**）。

2. 运行以下命令来创建对象：

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

2.9. 配置 INGRESS CONTROLLER

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。

前提条件

- 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。

2.9.2. 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。

前提条件

- 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 Red Hat OpenShift Service on AWS 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
```

2.9.3. 自动扩展 Ingress Controller

自动缩放 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 **IngressController** 的示例。

前提条件

1. 已安装 OpenShift CLI (**oc**)。
2. 您可以使用具有 **cluster-admin** 角色的用户访问 Red Hat OpenShift Service on AWS 集群。
3. 已安装自定义 Metrics Autoscaler Operator。
4. 您位于 **openshift-ingress-operator** 项目命名空间中。

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 serviceaccount thanos && oc describe serviceaccount thanos
```

输出示例

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-b4l9s
Mountable secrets:   thanos-dockercfg-b4l9s
Tokens:              thanos-token-c422q
Events:              <none>
```

2. 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. 运行以下命令，定义包含 **secret** 的变量 secret：

```
$ secret=$(oc get secret | grep thanos-token | head -n 1 | awk '{ print $1 }')
```

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

```
$ oc process TOKEN="$secret" -f - <<EOF | oc apply -f -
apiVersion: template.openshift.io/v1
kind: Template
parameters:
- name: TOKEN
objects:
- apiVersion: keda.sh/v1alpha1
  kind: TriggerAuthentication
  metadata:
```

```
    name: keda-trigger-auth-prometheus
  spec:
    secretTargetRef:
    - parameter: bearerToken
      name: \${TOKEN}
      key: token
    - parameter: ca
      name: \${TOKEN}
      key: ca.crt
EOF
```

3. 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
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
```

4. 输入以下命令在服务帐户中添加新角色：

```
$ oc adm policy 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

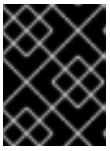
只有在使用跨命名空间查询时，才需要参数 **add-cluster-role-to-user**。以下步骤使用 **kube-metrics** 命名空间中的查询，该命名空间需要此参数。

5. 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
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。

6. 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 ingresscontroller/default -o yaml | grep replicas:
```

输出示例

```
replicas: 3
```

  - 获取 **openshift-ingress** 项目中的 pod：

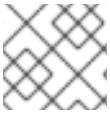
```
$ oc get pods -n openshift-ingress
```

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
```

2.9.4. 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。

2.9.5. 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。如果您有高流量集群，为了避免超过日志记录堆栈的容量，或者与 Red Hat OpenShift Service on AWS 以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。

前提条件

- 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**，则必须使用 **spec.logging.access.destination.syslog.endpoint** 指定一个目的地端点，并可使用 **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。

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
```

2.9.6. 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。

前提条件

- 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。

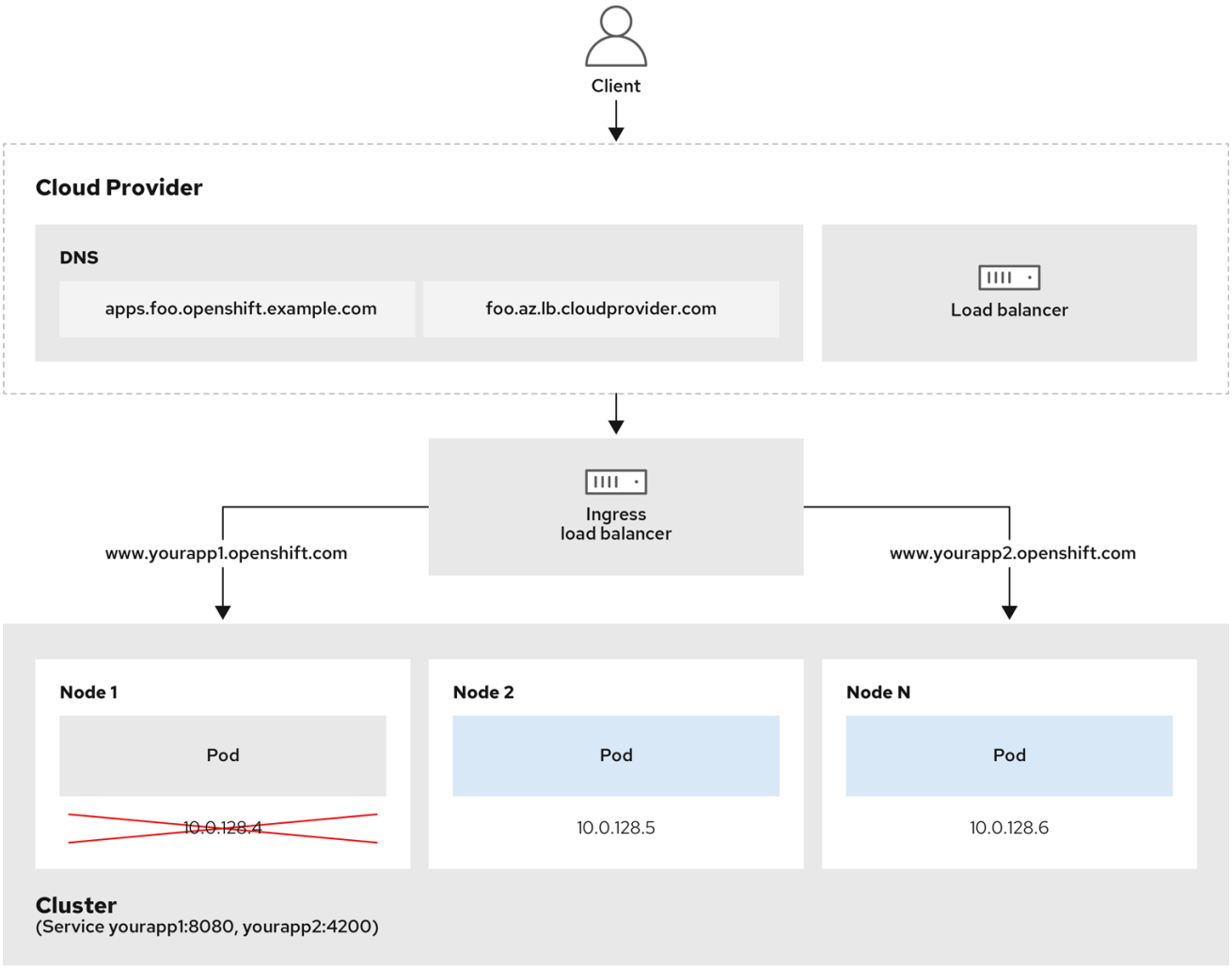
2.9.7. 配置 Ingress Controller 以使用内部负载均衡器

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 Red Hat OpenShift Service on AWS 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 。

前提条件

- 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
```

2.9.8. 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。

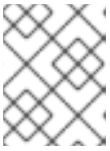
前提条件

- 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"}}}'
```

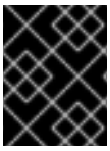


注意

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

2.9.9. 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。

前提条件

- 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
```

2.9.10. 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。

前提条件

- 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
```

2.9.11. 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
```

2.9.12. HTTP 标头配置

Red Hat OpenShift Service on AWS 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**。

2.9.12.1. 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。

2.9.12.2. 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
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

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
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

2.9.13. 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 mutual TLS，这需要您的应用程序检查 X-Forwarded-Client-Cert 请求标头，但 Red Hat OpenShift Service on AWS 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。

前提条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问 Red Hat OpenShift Service on AWS 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。

2.9.14. 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。

2.9.15. 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"
```

2.9.16. 为 Ingress Controller 配置 PROXY 协议

当 Ingress Controller 使用 **HostNetwork** 或 **NodePortService** 端点发布策略类型时，集群管理员可配置 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。

云部署不支持此功能。这个限制的原因是，当 AWS 上的 Red Hat OpenShift Service 在云平台中运行，IngressController 指定使用服务负载均衡器时，Ingress Operator 会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 Red Hat OpenShift Service on AWS 和外部负载均衡器配置为使用 PROXY 协议或使用 TCP。

警告

在使用 Keepalived Ingress VIP 的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。

前提条件

- 已创建一个 Ingress Controller。

流程

1. 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，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，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
```

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 Red Hat OpenShift Service on AWS 使用的可选域，而不是默认值，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。

前提条件

- 您已在 AWS 集群上部署了 Red Hat OpenShift Service。
- 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
   ```

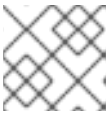
   b. 编辑 YAML 文件：

   示例 appsDomain 配置为 test.example.com

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

   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 Red Hat OpenShift Service on AWS 基础架构的域。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
   ```

2.9.18. 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

因为 Red Hat OpenShift Service on AWS 包含 HAProxy 2.8，因此请务必在升级前使用 **spec.httpHeaders.headerNameCaseAdjustments** 添加必要的配置。

前提条件

- 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

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。

- 使用 **oc patch** 命令设置一个 HTTP 标头的大小写情况。
  1. 输入 **oc patch** 命令将 HTTP **host** 标头改为 **Host**：

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

  2. 注解应用程序的路由：

     ```
     $ oc annotate routes/my-application haproxy.router.openshift.io/h1-adjust-case=true
     ```

     然后，Ingress Controller 会根据指定调整 **host** 请求标头。
- 通过配置 Ingress Controller YAML 文件，使用 **HeaderNameCaseAdjustments** 字段指定调整。
  1. 以下 Ingress Controller YAML 示例将 HTTP/1 请求的 **host** 标头调整为 **Host**，以便可以适当地注解路由：

     Ingress Controller YAML 示例

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

  2. 以下示例路由中，使用 **haproxy.router.openshift.io/h1-adjust-case** 注解启用对 HTTP 响应标头名称的大小写调整：

     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。

2.9.19. 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"
     ...
```

2.9.20. 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。

前提条件

- 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. 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",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. 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
```

2.9.21. 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 Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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。

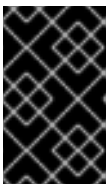
注意

如果您使用 Red Hat OpenShift Service on AWS 默认 503 错误代码页面作为自定义的模板，文件中的标头需要一个可以使用 CRLF 行结尾的编辑器。

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
```

2.9.22. 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。

前提条件

- 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。

2.10. RED HAT OPENSHIFT SERVICE ON AWS INGRESS OPERATOR 配置

下表详细介绍了 Ingress Operator 的组件，以及红帽站点可靠性工程师 (SRE) 是否维护 Red Hat OpenShift Service on AWS 中的这个组件。

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 AWS LOAD BALANCER OPERATOR

AWS Load Balancer Operator (ALBO)是红帽支持的 Operator，用户可以选择性地在 SRE 管理的 Red Hat OpenShift Service on AWS (ROSA)集群上安装。ALBO 管理 AWS 管理的 AWS Load Balancer Controller (ALBC)的生命周期，它为在 ROSA 集群中运行的应用程序置备 AWS Elastic Load Balancing v2 (ELBv2)服务。

3.1. 安装 AWS LOAD BALANCER OPERATOR

如果满足某些要求，您可以安装 AWS Load Balancer Operator (ALBO)。

先决条件

- 您有一个现有的 Red Hat OpenShift Service on AWS (ROSA)集群，且在多个可用区(AZ)模式下安装有 bring-your-own-VPC (BYO-VPC)配置。
- 您可以使用具有 **dedicated-admin** 角色的用户访问集群。
- 您可以访问创建的 ROSA 集群的 VPC 和子网。
- 已安装 ROSA CLI (**rosa**)。
- 已安装 Amazon Web Services (AWS) CLI。
- 已安装 OpenShift CLI（oc）。
- 您使用 OpenShift Container Platform (OCP) 4.13 或更高版本。

重要

当安装 ALBO 以用于 AWS Local Zone (LZ)中的 ROSA 集群时，您必须为帐户启用 AWS LZ，并且 AWS Elastic Load Balancing v2 (ELBv2)服务必须在 AWS LZ 中提供。

流程

1. 运行以下命令，识别集群基础架构 ID 和集群 OpenID Connect (OIDC) DNS：
   a. 确定 ROSA 集群 INFRA ID：

```
$ rosa describe cluster --cluster=<cluster_name> | grep -i 'Infra ID'
```

   或

```
$ oc get infrastructure cluster -o json | jq -r '.status.infrastructureName'
```

   b. 识别 ROSA 集群 OIDC DNS:

```
$ rosa describe cluster --cluster=<cluster_name> | grep -i 'OIDC'
```

   保存命令的输出。您将在此过程中在以后的步骤中使用此信息。

2. 创建 ALBO 所需的 AWS IAM 策略：
   a. 以具有 **dedicated-admin** 角色的用户身份登录 ROSA 集群，并使用以下命令创建新项目：

```
$ oc new-project aws-load-balancer-operator
```

b. 将以下信任策略分配给新创建的 AWS IAM 角色：

```
$ IDP='{Cluster_OIDC_Endpoint}'
$ IDP_ARN="arn:aws:iam::{AWS_AccountNo}:oidc-provider/${IDP}" 1
$ cat <<EOF > albo-operator-trusted-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "${IDP_ARN}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${IDP}:sub": "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager"
                }
            }
        }
    ]
}
EOF
```

1 将 '{AWS_AccountNo}' 替换为您的 AWS 帐户号，将 '{Cluster_OIDC_Endpoint}' 替换为此流程前面标识的 OIDC DNS。

重要

在将 **{Cluster_OIDC_Endpoint}** 替换为之前标识的 OIDC DNS 时，不要包含 OIDC DNS URL 的 **https** 部分。只需要在 URL 中遵循 / 的字母数字信息。

有关为 AWS IAM 角色分配信任策略的更多信息，请参阅 如何将信任策略与 IAM 角色一起使用。

c. 使用生成的信任策略创建并验证角色：

```
$ aws iam create-role --role-name albo-operator --assume-role-policy-document file://albo-operator-trusted-policy.json
$ OPERATOR_ROLE_ARN=$(aws iam get-role --role-name albo-operator --output json | jq -r '.Role.Arn')
$ echo $OPERATOR_ROLE_ARN
```

有关创建 AWS IAM 角色的更多信息，请参阅创建 IAM 角色。

d. 将 Operator 的权限策略附加到角色：

```
$ curl -o albo-operator-permission-policy.json https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/release-1.1/hack/operator-permission-policy.json
```

```
$ aws iam put-role-policy --role-name albo-operator --policy-name perms-policy-albo-operator --policy-document file://albo-operator-permission-policy.json
```

有关在 AWS IAM 角色中添加 AWS IAM 权限的更多信息，请参阅 添加和删除 IAM 身份权限。

e. 生成 Operator 的 AWS 凭证：

```
$ cat <<EOF> albo-operator-aws-credentials.cfg
[default]
sts_regional_endpoints = regional
role_arn = ${OPERATOR_ROLE_ARN}
web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
EOF
```

有关格式化凭证文件的更多信息，请参阅 在 Amazon Web Services Security Token Service 中使用手动模式。

f. 使用生成的 AWS 凭证创建 Operator 凭证 secret：

```
$ oc -n aws-load-balancer-operator create secret generic aws-load-balancer-operator --from-file=credentials=albo-operator-aws-credentials.cfg
```

3. 创建 AWS Load Balancer Controller (ALBC)所需的 AWS IAM 策略：

a. 为您的身份提供程序生成信任策略文件。以下示例使用 OpenID Connect：

```
$ IDP='{Cluster_OIDC_Endpoint}'
$ IDP_ARN="arn:aws:iam::{AWS_AccountNo}:oidc-provider/${IDP}"
$ cat <<EOF > albo-controller-trusted-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "${IDP_ARN}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${IDP}:sub": "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-controller-cluster"
                }
            }
        }
    ]
}
EOF
```

b. 使用生成的信任策略创建并验证角色：

```
$ aws iam create-role --role-name albo-controller --assume-role-policy-document file://albo-controller-trusted-policy.json
$ CONTROLLER_ROLE_ARN=$(aws iam get-role --role-name albo-controller --output
```

```
json | jq -r '.Role.Arn')
$ echo $CONTROLLER_ROLE_ARN
```

c. 将控制器的权限策略附加到角色：

```
$ curl -o albo-controller-permission-policy.json
https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.4.7/docs/install/iam_policy.json
$ aws iam put-role-policy --role-name albo-controller --policy-name perms-policy-albo-
controller --policy-document file://albo-controller-permission-policy.json
```

d. 生成控制器的 AWS 凭证：

```
$ cat <<EOF > albo-controller-aws-credentials.cfg
[default]
sts_regional_endpoints = regional
role_arn = ${CONTROLLER_ROLE_ARN}
web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
EOF
```

e. 使用生成的 AWS 凭证创建控制器的凭证 secret：

```
$ oc -n aws-load-balancer-operator create secret generic aws-load-balancer-controller-
cluster --from-file=credentials=albo-controller-aws-credentials.cfg
```

4. 为子网发现添加所需的标签：

a. 将以下 **{Key: Value}** 标签添加到托管 ROSA 集群的 VPC 中以及所有子网中。将 **{Cluster Infra ID}** 替换为之前指定的 Infra ID:

```
* kubernetes.io/cluster/${Cluster Infra ID}:owned
```

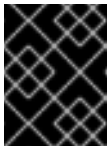
b. 将以下 ELBv2 **{Key: Value}** 标签添加到专用子网，也可以选择性地添加到公共子网中：

- 专用子网： **kubernetes.io/role/internal-elb:1**
- 公共子网： **kubernetes.io/role/elb:1**

注意

面向互联网和内部负载均衡器将在这些子网所属的 AZ 中创建。

有关向 AWS 资源添加标签（包括 VPC 和子网）的更多信息，请参阅 标记 Amazon EC2 资源。



重要

ELBv2 资源（如 ALBO 创建的 ALB 和 NLBs）不会继承为 ROSA 集群设置的自定义标签。您必须单独为这些资源设置标签。

5. 创建 ALBO：

```
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  upgradeStrategy: Default
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1.0
  installPlanApproval: Automatic
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: aws-load-balancer-operator.v1.0.0
```

6. 创建 AWS ALBC：

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Manual
  credentials:
    name: aws-load-balancer-controller-cluster
```

重要

因为 AWS ALBC 不支持创建与 AZ 和 AWS LZs 关联的 ALB，所以 ROSA 集群可以完全与 AWS LZ 或 AZ 关联，但不能同时创建 ALB。

有关 AWS ALBC 配置的更多信息，请参阅以下主题：

- 创建多个入口
- 添加 TLS 终止

验证

- 运行以下命令来确认安装成功：
  1. 收集项目中 pod 的信息：

```
$ oc get pods -n aws-load-balancer-operator
```

  2. 查看项目中的日志：

```
$ oc logs -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager -c manager
```

有关验证为 ROSA 集群中运行的应用程序创建了 ELBv2 的详细信息，请参阅创建 AWS Load Balancer Controller 实例。

3.2. 卸载 AWS LOAD BALANCER OPERATOR

要卸载 AWS Load Balancer Operator (ALBO)，并对相关资源执行总体清理，请执行以下步骤。

流程

1. 通过删除由 ALBO 创建和管理的 Load Balancers 来清理示例应用程序。有关删除负载均衡器的更多信息，请参阅 删除应用程序负载均衡器。
2. 删除添加到子网中的 VPC 标签，以及创建 Application Load Balancers (ALBs)来清理 AWS VPC 标签。如需更多信息，请参阅 标签基础知识。
3. 通过删除 ALBO 和 Application Load Balancer Controller (ALBC)来清理 ALBO 组件。如需更多信息，请参阅 从集群中删除 Operator。

第 4 章 OPENSHIFT SDN 默认 CNI 网络供应商

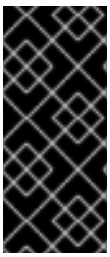
4.1. 为项目启用多播

注意

从 Red Hat OpenShift Service on AWS 4.14 开始，OpenShift SDN CNI 已被弃用。从 Red Hat OpenShift Service on AWS 4.15 开始，网络插件不是新安装的选项。在以后的发行版本中，计划删除 OpenShift SDN 网络插件，并不再被支持。红帽将在删除前对这个功能提供程序错误修正和支持，但不会再改进这个功能。作为 OpenShift SDN CNI 的替代选择，您可以使用 OVN Kubernetes CNI。

4.1.1. 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，Red Hat OpenShift Service on AWS 间的多播流量被禁用。如果使用 OpenShift SDN 网络插件，可以根据每个项目启用多播。

在 **networkpolicy** 隔离模式中使用 OpenShift SDN 网络插件：

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

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

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

4.1.2. 启用 pod 间多播

您可以为项目启用 pod 间多播。

先决条件

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

流程

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

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

验证

要验证项目是否启用了多播，请完成以下步骤：

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

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

2. 创建 pod 以作为多播接收器：

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/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：

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 5 章 ROSA 集群的网络验证

当您将 Red Hat OpenShift Service on AWS (ROSA)集群部署到现有的 Virtual Private Cloud (VPC)中，或使用对集群的新子网创建额外的机器池时，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。

5.1. 了解 ROSA 集群的网络验证

当您将 Red Hat OpenShift Service on AWS (ROSA)集群部署到现有的 Virtual Private Cloud (VPC)中时，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。如果您使用带有互动模式的 ROSA CLI (**rosa**)创建集群，则检查会在提供所需的 VPC 网络信息后运行。如果您在没有互动模式的情况下使用 CLI，则检查会在集群创建前立即开始。

当您添加带有集群新子网的机器池时，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，并且部署具有更大的成功机会。

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*。

5.2. 网络验证检查的范围

网络验证包括以下每个要求：

- 父虚拟私有云(VPC)存在。
- 所有指定子网都属于 VPC。
- VPC 已启用 **enableDnsSupport**。
- VPC 已启用 **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。

5.3. 绕过自动网络验证

如果要将带有已知网络配置问题的 Red Hat OpenShift Service on AWS (ROSA)集群部署到现有的 Virtual Private Cloud (VPC)集群中，您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，集群具有有限的支持状态。安装后，您可以解决这个问题，然后手动运行网络验证。验证成功后会删除有限的支持状态。

使用 OpenShift Cluster Manager 绕过自动网络验证

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 验证** 来绕过自动验证。

5.4. 手动运行网络验证

在 AWS (ROSA)集群上安装 Red Hat OpenShift Service 后，您可以使用 Red Hat OpenShift Cluster Manager 或 ROSA CLI (**rosa**)手动运行网络验证检查。

使用 OpenShift Cluster Manager 手动运行网络验证

您可以使用 Red Hat OpenShift Cluster Manager 手动在 AWS (ROSA)集群上运行网络验证检查。

先决条件

- 您有一个现有的 ROSA 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**。

使用 CLI 手动运行网络验证

您可以使用 ROSA CLI (**rosa**)手动在 AWS (ROSA)集群上运行网络验证检查。

运行网络验证时，您可以指定一组 VPC 子网 ID 或集群名称。

先决条件

- 您已在安装主机上安装并配置了最新的 ROSA CLI (**rosa**)。
- 您有一个现有的 ROSA 集群。
- 您是集群所有者，或具有集群编辑器角色。

流程

- 使用以下方法之一验证网络配置：
  - 通过指定集群名称来验证网络配置。子网 ID 会自动检测到：

```
$ rosa verify network --cluster <cluster_name> 1
```

1 将 **<cluster_name>** 替换为集群的名称。

输出示例

```
I: Verifying the following subnet IDs are configured correctly: [subnet-
03146b9b52b6024cb subnet-03146b9b52b2034cc]
I: subnet-03146b9b52b6024cb: pending
I: subnet-03146b9b52b2034cc: passed
I: Run the following command to wait for verification to all subnets to complete:
rosa verify network --watch --status-only --region us-east-1 --subnet-ids subnet-
03146b9b52b6024cb,subnet-03146b9b52b2034cc
```

    - 确保所有子网的验证已完成：

```
$ rosa verify network --watch \ 1
                      --status-only \ 2
                      --region <region_name> \ 3
                      --subnet-ids subnet-03146b9b52b6024cb,subnet-03146b9b52b2034cc
4
```

1. **watch** 标志会在测试下的所有子网都处于失败或传递状态后完成。
2. **status-only** 标志不会触发网络验证运行，而是返回当前状态，如 **subnet-123 （仍然在进行中验证）**。默认情况下，如果没有这个选项，对此命令的调用始终会触发对指定子网的验证。
3. 使用覆盖 *AWS_REGION* 环境变量的特定 AWS 区域。
4. 输入以逗号分开的子网 ID 列表进行验证。如果有任何子网不存在，则显示子网 **'subnet-<subnet_number> not found 的错误消息网络**验证信息，且不会检查子网。

输出示例

```
I: Checking the status of the following subnet IDs: [subnet-03146b9b52b6024cb
subnet-03146b9b52b2034cc]
I: subnet-03146b9b52b6024cb: passed
I: subnet-03146b9b52b2034cc: passed
```

提示

要输出完整的验证测试列表，您可以在运行 **rosa verify network** 命令时包含 **--debug** 参数。

- 通过指定 VPC 子网 ID 来验证网络配置。将 **<region_name&** gt; 替换为您的 AWS 区域，将 **<AWS_account_ID** > 替换为您的 AWS 帐户 ID：

```
$ rosa verify network --subnet-ids 03146b9b52b6024cb,subnet-03146b9b52b2034cc --
region <region_name> --role-arn arn:aws:iam::<AWS_account_ID>:role/my-Installer-
Role
```

输出示例

```
I: Verifying the following subnet IDs are configured correctly: [subnet-
03146b9b52b6024cb subnet-03146b9b52b2034cc]
I: subnet-03146b9b52b6024cb: pending
I: subnet-03146b9b52b2034cc: passed
I: Run the following command to wait for verification to all subnets to complete:
rosa verify network --watch --status-only --region us-east-1 --subnet-ids subnet-
03146b9b52b6024cb,subnet-03146b9b52b2034cc
```

- 确保所有子网的验证已完成：

```
$ rosa verify network --watch --status-only --region us-east-1 --subnet-ids subnet-
03146b9b52b6024cb,subnet-03146b9b52b2034cc
```

输出示例

```
I: Checking the status of the following subnet IDs: [subnet-03146b9b52b6024cb
subnet-03146b9b52b2034cc]
I: subnet-03146b9b52b6024cb: passed
I: subnet-03146b9b52b2034cc: passed
```

第 6 章 配置集群范围代理

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

注意

只有集群系统出口流量会被代理，包括对云供应商 API 的调用。

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。

6.1. 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)。
- 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

- 如果您的代理重新加密出口流量，则必须对域和端口组合创建排除。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。

其它资源

- 有关使用 AWS 安全令牌服务(STS)的 ROSA 集群 的安装先决条件，请参阅使用 STS 的 ROSA 的 AWS 先决条件。

- 有关不使用 STS 的 ROSA 集群的安装先决条件，请参阅 ROSA 的 AWS 先决条件。

6.2. 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

6.3. 在安装过程中配置代理

当您在 AWS (ROSA) 集群中安装 Red Hat OpenShift Service (ROSA) 集群时，可以配置 HTTP 或 HTTPS 代理到现有的 Virtual Private Cloud (VPC)集群中。您可以使用 Red Hat OpenShift Cluster Manager 或 ROSA CLI (**rosa**) 在安装过程中配置代理。

6.3.1. 使用 OpenShift Cluster Manager 在安装过程中配置代理

如果要在 AWS (ROSA) 集群上安装 Red Hat OpenShift Service，您可以在安装过程中使用 Red Hat OpenShift Cluster Manager 启用集群范围的 HTTP 或 HTTPS 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*使用 OpenShift Cluster Manager 使用自定义创建集群*。

6.3.2. 使用 CLI 在安装过程中配置代理

如果要在 AWS (ROSA) 集群中安装 Red Hat OpenShift Service (ROSA) 到现有的 Virtual Private Cloud (VPC)，您可以使用 ROSA CLI (**rosa**) 在安装过程中启用集群范围的 HTTP 或 HTTPS 代理。

以下流程提供有关在安装过程中配置集群范围代理的 ROSA CLI (**rosa**) 参数的详细信息。有关使用 ROSA CLI 的常规安装步骤，请参阅*使用 CLI 创建自定义集群*。

先决条件

- 您已确认代理可以从安装集群的 VPC 访问。该代理还必须从 VPC 的专用子网访问。

流程

- 指定创建集群时的代理配置：

```
$ rosa create cluster \
 <other_arguments_here> \
 --additional-trust-bundle-file <path_to_ca_bundle_file> \ 1 2 3
 --http-proxy http://<username>:<password>@<ip>:<port> \ 4 5
 --https-proxy https://<username>:<password>@<ip>:<port> \ 6 7
 --no-proxy example.com 8
```

1 4 6 **additional-trust-bundle-file**、**http-proxy** 和 **https-proxy** 参数都是可选。

2 **additional-trust-bundle-file** 参数是一个指向 PEM 编码 X.509 证书的捆绑包的文件路径，这些证书全部连接在一起。使用 TLS-inspecting 代理的用户需要 additional-trust-bundle-file 参数，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 参数明确配置，都适用。

3 5 7 **http-proxy** 和 **https-proxy** 参数必须指向有效的 URL。

8 要排除代理的目标域名、IP 地址或网络 CIDR 的逗号分隔列表。

在域前面加上 **.** 以仅匹配子域。例如： **.y.com** 匹配 **x.y.com**，但不匹配 **y.com**。使用 * 可对所有目的地绕过所有代理。如果您扩展了未包含在安装配置中 **networking.machineNetwork[].cidr** 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

如果未设置 **httpProxy** 或 **httpsProxy** 字段，则此字段将被忽略。

其它资源

- 使用 OpenShift Cluster Manager 使用自定义创建集群
- 使用 CLI 使用自定义创建集群

6.4. 安装后配置代理

当您在 AWS (ROSA) 集群中安装 Red Hat OpenShift Service (ROSA) 集群后，可以配置 HTTP 或 HTTPS 代理到现有的 Virtual Private Cloud (VPC) 集群中。您可以使用 Red Hat OpenShift Cluster Manager 或 ROSA CLI (**rosa**) 在安装后配置代理。

6.4.1. 使用 OpenShift Cluster Manager 在安装后配置代理

您可以使用 Red Hat OpenShift Cluster Manager 将集群范围代理配置添加到 Virtual Private Cloud (VPC) 上的 AWS 集群中的现有 Red Hat OpenShift Service 中。

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。

前提条件

- 在 AWS 集群上有一个 Red Hat OpenShift Service。
- 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**。

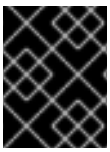
验证

- 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。

6.4.2. 使用 CLI 在安装后配置代理

您可以使用 AWS (ROSA) CLI (**rosa**) 上的 Red Hat OpenShift Service 将集群范围的代理配置添加到 Virtual Private Cloud (VPC) 的现有 ROSA 集群中。

您还可以使用 **rosa** 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。

前提条件

- 您已在安装主机上安装和配置了最新的 ROSA (**rosa**) 和 OpenShift (**oc**) CLI。
- 您有一个在 VPC 中部署的 ROSA 集群。

流程

- 编辑集群配置以添加或删除集群范围代理详情：

```
$ rosa edit cluster \
 --cluster $CLUSTER_NAME \
 --additional-trust-bundle-file <path_to_ca_bundle_file> \ 1 2 3
 --http-proxy http://<username>:<password>@<ip>:<port> \ 4 5
 --https-proxy https://<username>:<password>@<ip>:<port> \ 6 7
  --no-proxy example.com 8
```

1 4 6 **additional-trust-bundle-file**、**http-proxy** 和 **https-proxy** 参数都是可选。

2 **additional-trust-bundle-file** 参数是一个指向 PEM 编码 X.509 证书的捆绑包的文件路径，这些证书全部连接在一起。additional-trust-bundle-file 参数是一个指向 PEM 编码 X.509 证

注意

您不应该尝试在集群中直接更改代理或其他信任捆绑包配置。这些更改必须使用 ROSA CLI (**rosa**) 或 Red Hat OpenShift Cluster Manager 加以应用。直接对集群所做的任何更改都会被自动恢复。

3 5 7 **http-proxy** 和 **https-proxy** 参数必须指向有效的 URL。

8 要排除代理的目标域名、IP 地址或网络 CIDR 的逗号分隔列表。

在域前面加上 **.** 以仅匹配子域。例如： **.y.com** 匹配 **x.y.com**，但不匹配 **y.com**。使用 * 可对所有目的地绕过所有代理。如果您扩展了未包含在安装配置中 **networking.machineNetwork[].cidr** 字段定义的 worker，您必须将它们添加到此列表中，以防止连接问题。

如果未设置 **httpProxy** 或 **httpsProxy** 字段，则此字段将被忽略。

验证

1. 列出机器配置池的状态，并验证它们是否已更新：

```
$ oc get machineconfigpools
```

输出示例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-d9a03f612a432095dcde6dcf44597d90   True      False      False
3            3                 3                   0                    31h
worker   rendered-worker-f6827a4efe21e155c25c21b43c46f65e   True      False      False
6            6                 6                   0                    31h
```

2. 显示集群的代理配置，并验证详情是否如预期：

```
$ oc get proxy cluster -o yaml
```

输出示例

```
apiVersion: config.openshift.io/v1
kind: Proxy
spec:
  httpProxy: http://proxy.host.domain:<port>
  httpsProxy: https://proxy.host.domain:<port>
  <...more...>
status:
  httpProxy: http://proxy.host.domain:<port>
  httpsProxy: https://proxy.host.domain:<port>
  <...more...>
```

6.5. 删除集群范围代理

您可以使用 ROSA CLI 删除集群范围代理。删除集群后，您还应删除添加到集群的任何信任捆绑包。

6.5.1. 使用 CLI 删除集群范围代理

您必须使用 Red Hat OpenShift Service on AWS (ROSA) CLI **rosa** 从集群中删除代理地址。

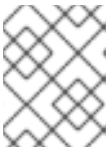
前提条件

- 您必须具有集群管理员特权。
- 已安装 ROSA CLI (**rosa**)。

流程

- 使用 **rosa edit** 命令修改代理。您必须将空字符串传递给 **--http-proxy** 和 **--https-proxy** 参数，才能清除集群中的代理：

```
$ rosa edit cluster -c <cluster_name> --http-proxy "" --https-proxy ""
```



注意

虽然代理可能只使用一个代理参数，但空字段将被忽略，因此将空字符串传递给 **--http-proxy** 和 **--https-proxy** 参数不会造成任何问题。

输出示例

```
I: Updated cluster <cluster_name>
```

验证

- 您可以使用 **rosa describe** 命令验证代理是否已从集群中删除：

```
$ rosa describe cluster -c <cluster_name>
```

在删除前，代理 IP 会显示在 proxy 部分：

```
Name:                  <cluster_name>
ID:                 <cluster_internal_id>
External ID:           <cluster_external_id>
OpenShift Version:       4.0
Channel Group:          stable
DNS:                <dns>
AWS Account:           <aws_account_id>
API URL:             <api_url>
Console URL:           <console_url>
Region:             us-east-1
Multi-AZ:           false
Nodes:
 - Control plane:       3
 - Infra:            2
```

```
 - Compute:              2
Network:
 - Type:                 OVNKubernetes
 - Service CIDR:         <service_cidr>
 - Machine CIDR:         <machine_cidr>
 - Pod CIDR:             <pod_cidr>
 - Host Prefix:          <host_prefix>
Proxy:
 - HTTPProxy:            <proxy_url>
Additional trust bundle:    REDACTED
```

删除代理后，代理部分会被删除：

```
Name:                   <cluster_name>
ID:                     <cluster_internal_id>
External ID:            <cluster_external_id>
OpenShift Version:      4.0
Channel Group:          stable
DNS:                    <dns>
AWS Account:            <aws_account_id>
API URL:                <api_url>
Console URL:            <console_url>
Region:                 us-east-1
Multi-AZ:               false
Nodes:
 - Control plane:       3
 - Infra:               2
 - Compute:             2
Network:
 - Type:                 OVNKubernetes
 - Service CIDR:         <service_cidr>
 - Machine CIDR:         <machine_cidr>
 - Pod CIDR:             <pod_cidr>
 - Host Prefix:          <host_prefix>
Additional trust bundle:    REDACTED
```

6.5.2. 删除 Red Hat OpenShift Service on AWS 集群上的证书颁发机构

您可以使用 Red Hat OpenShift Service on AWS (ROSA) CLI (**rosa**)从集群中删除证书颁发机构(CA)。

前提条件

- 您必须具有集群管理员特权。
- 已安装 ROSA CLI (**rosa**)。
- 集群添加了证书颁发机构。

流程

- 使用 **rosa edit** 命令修改 CA 信任捆绑包。您必须将空字符串传递给 **--additional-trust-bundle-file** 参数，以便从集群中清除信任捆绑包：

```
$ rosa edit cluster -c <cluster_name> --additional-trust-bundle-file ""
```

输出示例

```
I: Updated cluster <cluster_name>
```

验证

- 您可以使用 **rosa describe** 命令验证信任捆绑包是否已从集群中删除：

```
$ rosa describe cluster -c <cluster_name>
```

在删除前，会显示额外的信任捆绑包部分，以便将其值用于安全目的：

```
Name:                  <cluster_name>
ID:                    <cluster_internal_id>
External ID:           <cluster_external_id>
OpenShift Version:     4.0
Channel Group:         stable
DNS:                   <dns>
AWS Account:           <aws_account_id>
API URL:               <api_url>
Console URL:           <console_url>
Region:                us-east-1
Multi-AZ:              false
Nodes:
 - Control plane:      3
 - Infra:              2
 - Compute:            2
Network:
 - Type:               OVNKubernetes
 - Service CIDR:       <service_cidr>
 - Machine CIDR:       <machine_cidr>
 - Pod CIDR:           <pod_cidr>
 - Host Prefix:        <host_prefix>
Proxy:
 - HTTPProxy:          <proxy_url>
Additional trust bundle:  REDACTED
```

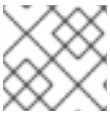
删除代理后，额外的信任捆绑包会被删除：

```
Name:                  <cluster_name>
ID:                    <cluster_internal_id>
External ID:           <cluster_external_id>
OpenShift Version:     4.0
Channel Group:         stable
DNS:                   <dns>
AWS Account:           <aws_account_id>
API URL:               <api_url>
Console URL:           <console_url>
Region:                us-east-1
Multi-AZ:              false
Nodes:
 - Control plane:      3
 - Infra:              2
 - Compute:            2
```

```
Network:
 - Type:                OVNKubernetes
 - Service CIDR:          <service_cidr>
 - Machine CIDR:            <machine_cidr>
 - Pod CIDR:              <pod_cidr>
 - Host Prefix:           <host_prefix>
Proxy:
 - HTTPProxy:             <proxy_url>
```

第 7 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 AWS 4.14 及更高版本上的 Red Hat OpenShift Service 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。**

7.1. 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。

注意

在使用带有 HCP 的 ROSA 时，静态 IP 地址 172.20.0.1 会为内部 Kubernetes API 地址保留。机器、pod 和服务 CIDR 范围不得与此 IP 地址冲突。

7.2. SERVICE CIDR

在 Service CIDR 字段中，您必须为服务指定 IP 地址范围。建议不要要求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。

7.3. POD CIDR

在 pod CIDR 字段中，您必须为 pod 指定 IP 地址范围。

建议不要要求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。

7.4. 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 8 章 网络安全性

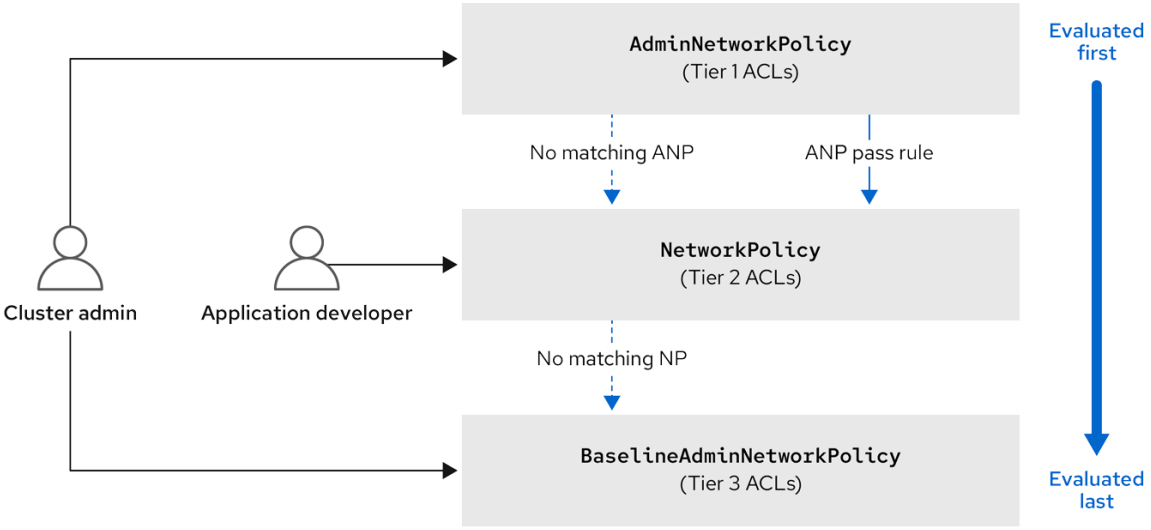
8.1. 了解网络策略 API

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。

Red Hat OpenShift Service on AWS 中的 OVN-Kubernetes CNI 使用访问控制列表(ACL) Tier 评估和应用它们来实施这些网络策略。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。



615_OpenShift_0324

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。

8.1.1. 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

8.2. 管理网络策略

8.2.1. OVN-Kubernetes AdminNetworkPolicy

8.2.1.1. AdminNetworkPolicy

AdminNetworkPolicy (ANP)是一个集群范围的自定义资源定义(CRD)。作为 Red Hat OpenShift Service on AWS 管理员，您可以在创建命名空间前创建网络策略来使用 ANP 来保护您的网络。另外，您可以在集群范围的级别上创建网络策略，该级别不可由 NetworkPolicy 对象覆盖。

AdminNetworkPolicy 和 NetworkPolicy 对象之间的关键区别在于，供管理员使用，是集群范围，而后者则用于租户所有者，并且是命名空间范围。

ANP 允许管理员指定以下内容：

- 确定其评估顺序的 priority 值。数值越低，优先级越高。
- 由应用策略的一组命名空间或命名空间组成的一组 pod。
- 要应用到 subject 的所有入口流量的入站规则列表。
- 用于来自 subject 的所有出口流量的出口规则列表。

AdminNetworkPolicy 示例

例 8.1. ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: sample-anp-deny-pass-rules (1)
spec:
  priority: 50 (2)
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name (3)
  ingress: (4)
  - name: "deny-all-ingress-tenant-1" (5)
    action: "Deny"
    from:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-anp: tenant-1
```

```
      podSelector:
        matchLabels:
          custom-anp: tenant-1 6
  egress:7
  - name: "pass-all-egress-to-tenant-1"
    action: "Pass"
    to:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-anp: tenant-1
        podSelector:
          matchLabels:
            custom-anp: tenant-1
```

1 为您的 ANP 指定一个名称。

2 spec.priority 字段支持集群中 0-99 值中最多 100 ANP。数值越低，优先级越高。创建具有相同优先级的 AdminNetworkPolicy 会创建非确定的结果。

3 指定要应用 ANP 资源的命名空间。

4 ANP 具有入口和出口规则。spec.ingress 字段的 ANP 规则接受 Pass,Deny, action 字段接受的值为 Allow。

5 为 ingress.name 指定一个名称。

6 指定 podSelector.matchLabels，以选择 namespaceSelector.matchLabels 作为入口对等选择的命名空间中的 pod。

7 ANPs 同时具有入口和出口规则。spec.egress 字段的 ANP 规则接受 Pass,Deny, action 字段接受的值为 Allow。

其他资源

- Network Policy API 工作组

8.2.1.1.1. 规则的 AdminNetworkPolicy 操作

作为管理员，您可以将您的 AdminNetworkPolicy 规则的 action 字段设置为 Allow,Deny, 或 Pass。由于 OVN-Kubernetes 使用分层 ACL 来评估网络流量规则，因此 3NP 允许您设置非常强大的策略规则，它们只能被管理员修改、删除规则，或通过设置更高优先级规则来覆盖它们。

AdminNetworkPolicy Allow 示例

在优先级 9 中定义的以下 ANP 可确保允许从 monitoring 命名空间到集群中的任何租户（所有其他命名空间）的所有入口流量。

例 8.2. 强 Allow ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: allow-monitoring
spec:
  priority: 9
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift namespaces as well.
  ingress:
  - name: "allow-ingress-from-monitoring"
    action: "Allow"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这是强的 Allow ANP 的示例，因为它不可以被涉及的所有方覆盖。租户都不会阻止自己被使用 NetworkPolicy 对象监控，监控租户也不知道它可以或无法监控的内容。

AdminNetworkPolicy 拒绝示例

在优先级 5 中定义的以下 ANP 可确保 monitoring 命名空间中的所有入口流量都被阻止到受限租户（具有标签 security: restricted的命名空间）。

例 8.3. 强 Deny ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
```

```
kind: AdminNetworkPolicy
metadata:
  name: block-monitoring
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        security: restricted
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这是一个强大的 Deny ANP，这是所有涉及的方都无法覆盖的。受限租户所有者无法授权自己允许监控流量，基础架构监控服务无法从这些敏感命名空间中提取任何内容。

与强的 Allow 示例结合使用时，block-monitoring ANP 具有较低优先级的值，赋予其优先级更高的优先级，这样可确保不会监控受限租户。

AdminNetworkPolicy Pass 示例

在优先级 7 定义的以下 ANP 可确保所有从 monitoring 命名空间到内部基础架构租户（具有标签 security: internal）的入口流量都将传递到 ACL 的层 2，并由命名空间的 NetworkPolicy 对象评估。

例 8.4. 强 Pass ANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: pass-monitoring
spec:
  priority: 7
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "pass-ingress-from-monitoring"
    action: "Pass"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

这个示例是一个强大的 Pass 操作 ANP，因为它将决策委派给租户所有者定义的 NetworkPolicy 对象。如果基础架构监控服务应使用命名空间范围 NetworkPolicy 对象提取其指标，则此 pass-monitoring ANP 允许在安全级别 internal 分组的所有租户所有者。

8.2.2. OVN-Kubernetes BaselineAdminNetworkPolicy

8.2.2.1. BaselineAdminNetworkPolicy

BaselineAdminNetworkPolicy (BANP)是一个集群范围的自定义资源定义(CRD)。作为 Red Hat OpenShift Service on AWS 管理员，您可以使用 BANP 设置并强制实施可选的基准网络策略规则，以便在需要时使用 NetworkPolicy 对象覆盖和强制实施使用 NetworkPolicy 对象的用户覆盖。BANP 的规则操作是 allow 或 deny。

BaselineAdminNetworkPolicy 资源是一个集群单例对象，当传递的流量策略与集群中的任何 NetworkPolicy 对象不匹配时，可用作 guardrail 策略。BANP 也可以用作默认安全模型，该模型默认阻止集群内流量，用户需要使用 NetworkPolicy 对象来允许已知的流量。在创建 BANP 资源时，必须使用 default 作为名称。

管理员可通过 BANP 指定：

- 由一组命名空间或命名空间的 subject。
- 要应用到 subject 的所有入口流量的入站规则列表。
- 用于来自 subject 的所有出口流量的出口规则列表。

BaselineAdminNetworkPolicy 示例

例 8.5. BANP 的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default 1
spec:
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name 2
  ingress: 3
```

```
  - name: "deny-all-ingress-from-tenant-1" 4
    action: "Deny"
    from:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-banp: tenant-1 5
        podSelector:
          matchLabels:
            custom-banp: tenant-1 6
  egress:
  - name: "allow-all-egress-to-tenant-1"
    action: "Allow"
    to:
    - pods:
        namespaceSelector:
          matchLabels:
            custom-banp: tenant-1
        podSelector:
          matchLabels:
            custom-banp: tenant-1
```

1 策略名称必须是 default，因为 BANP 是一个单例对象。

2 指定要将 ANP 应用到的命名空间。

3 BANP 具有入口和出口规则。spec.ingress 和 spec.egress 字段的 BANP 规则接受 Deny, action 字段接受的值为 Allow。

4 为 ingress.name 指定名称

5 指定要从中选择 pod 以应用 BANP 资源的命名空间。

6 指定 podSelector.matchLabels 名称，以应用 BANP 资源。

BaselineAdminNetworkPolicy 拒绝示例

以下 BANP 单例确保管理员为 internal 安全级别进入租户的所有入口监控流量设置了默认的拒绝策略。与 "AdminNetworkPolicy Pass example" 组合时，这个 deny 策略充当 ANP pass-monitoring 策略传递的所有入口流量的保护策略。

例 8.6. guardrail Deny 规则的 YAML 文件示例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

您可以将带有 action 字段的值为 Pass 的 AdminNetworkPolicy 资源与 BaselineAdminNetworkPolicy 资源结合使用来创建多租户策略。此多租户策略允许一个租户在应用上收集监控数据，同时不从第二个租户收集数据。

作为管理员，如果您同时应用了 "AdminNetworkPolicy Pass action example" 和 "BaselineAdminNetwork Policy Deny example"，则租户将保留创建在 BANP 之前评估的 NetworkPolicy 资源。

例如，租户 1 可以设置以下 NetworkPolicy 资源来监控入口流量：

例 8.7. NetworkPolicy 示例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-monitoring
  namespace: tenant 1
spec:
  podSelector:
  policyTypes:
    - Ingress
  ingress:
```

```
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

在这种情况下，Tenant 1 会在 "AdminNetworkPolicy Pass action example" 之后，"BaselineAdminNetwork Policy Deny example" 之前被评估，它将拒绝所有进入 安全级别 internal 的入口监控流量。随着租户 1 的 NetworkPolicy 对象就位，它们将能够在其应用程序中收集数据。但是，租户 2 没有任何 NetworkPolicy 对象，将无法收集数据。作为管理员，您没有默认监控内部租户，而是创建了 BANP，它允许租户使用 NetworkPolicy 对象覆盖 BANP 的默认行为。

8.3. 网络策略

8.3.1. 关于网络策略

作为集群管理员，您可以定义网络策略以限制到集群中的 pod 的网络通讯。

8.3.1.1. 关于网络策略

在使用支持 Kubernetes 网络策略的网络插件的集群中，网络隔离完全由 NetworkPolicy 对象控制。在 AWS 4 上的 Red Hat OpenShift Service 中，OpenShift SDN 支持在默认的网络隔离模式中使用网络策略。

警告

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

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。

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 AWS Ingress Controller 上的 Red Hat OpenShift Service 的连接：

  要使项目只允许 AWS Ingress Controller 上的 Red Hat OpenShift Service 的连接，请添加以下 NetworkPolicy 对象。

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

- 只接受项目中 pod 的连接：

  要使 pod 接受同一项目中其他 pod 的连接，但拒绝其他项目中所有 pod 的连接，请添加以

下 NetworkPolicy 对象：

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

- 仅允许基于 pod 标签的 HTTP 和 HTTPS 流量：

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- 使用命名空间和 pod 选择器接受连接：

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

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
```

```
    matchLabels:
      project: project_name
  podSelector:
    matchLabels:
      name: test-pods
```

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

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

8.3.1.1.1. 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: "" 1
  podSelector: {}
  policyTypes:
  - Ingress
```

1

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

8.3.1.1.2. 使用 allow-from-hostnetwork 网络策略

添加以下 allow-from-hostnetwork NetworkPolicy 对象来指示来自主机网络 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
```

8.3.1.2. 使用 OpenShift SDN 优化网络策略

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

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

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

在设计您的网络策略时，请参考以下指南：

- 使用命名空间使其包含需要隔离的 pod 组，可以减少 OVS 流规则数量。

  使用 namespaceSelector 或空 podSelector 选择整个命名空间的NetworkPolicy 对象会只生成 一个与命名空间的 VXLAN 虚拟网络 ID（VNID）匹配的 OVS 流量规则。

- 保留不需要在原始命名空间中隔离的 pod，并将需要隔离的 pod 移到一个或多个不同的命名空间中。

- 创建额外的目标跨命名空间网络策略，以允许来自不同隔离的 pod 的特定流量。

8.3.1.3. 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。

8.3.1.4. 后续步骤

- 创建网络策略

8.3.2. 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。

8.3.2.1. 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。

8.3.2.2. 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。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略规则：

   a. 创建一个 <policy_name>.yaml 文件：

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

   其中：

   <policy_name>

   指定网络策略文件名。

   b. 在您刚才创建的文件中定义网络策略，如下例所示：

   拒绝来自所有命名空间中的所有 pod 的入口流量

   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>

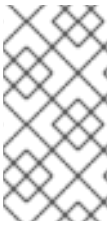
指定网络策略文件名。

<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。

8.3.2.3. 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。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建以下 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
```

8.3.2.4. 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。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建策略，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:
    - {}
```

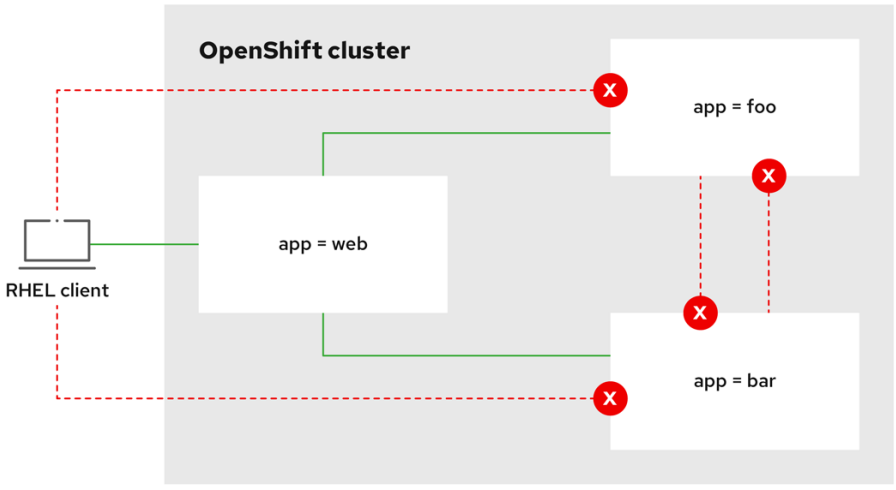
2. 输入以下命令应用策略：

```
$ 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

8.3.2.5. 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 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建一个策略，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. 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>
```

8.3.2.6. 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 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略要应用到的命名空间中。

流程

1. 创建一个策略，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>
```

8.3.2.7. 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。

前提条件

- 已登陆到 OpenShift Cluster Manager。
- 您在 AWS 集群上创建了 Red Hat OpenShift Service。
- 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 AWS 集群上的 Red Hat OpenShift Service 中创建项目。

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。

8.3.3. 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。

8.3.3.1. 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

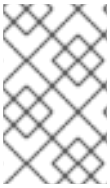
与策略对象允许从中入口流量的 pod 匹配的选择器。选择器与 NetworkPolicy 在同一命名空间中的 pod 匹配。

4

接受流量的一个或多个目标端口的列表。

8.3.3.2. 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。

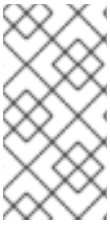
例如：

```
$ 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。

8.3.3.3. 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。
- 您在 AWS 集群上创建了 Red Hat OpenShift Service。
- 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。

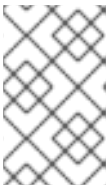
流程

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

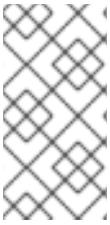
您只能查看这些策略的详情。您不能编辑这些策略。

8.3.4. 编辑网络策略

作为具有 admin 角色的用户，您可以编辑命名空间的现有网络策略。

8.3.4.1. 编辑网络策略

您可以编辑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，则可以在集群中的任何命名空间中编辑网络策略。

前提条件

- 集群使用支持 NetworkPolicy 对象的网络插件，如 OVN-Kubernetes 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。
- 您在网络策略所在的命名空间中。

流程

1. 可选：要列出一个命名空间中的网络策略对象，请输入以下命令：

```
$ oc get networkpolicy
```

其中：

<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。

2. 编辑网络策略对象。

- 如果您在文件中保存了网络策略定义，请编辑该文件并进行必要的更改，然后输入以下命令。

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

其中：

<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。

<policy_file>

指定包含网络策略的文件的名称。

- 如果您需要直接更新网络策略对象，请输入以下命令：

```
$ oc edit 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。

3. 确认网络策略对象已更新。

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

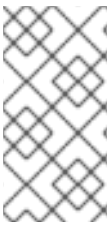
其中：

<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。



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 或通过 Actions 菜单从 web 控制台中的策略编辑网络策略。

8.3.4.2. 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。

8.3.4.3. 其他资源

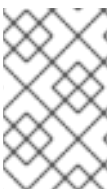
- 创建网络策略

8.3.5. 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。

8.3.5.1. 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 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。

- 已安装 OpenShift CLI（oc）。

- 您可以使用具有 admin 权限的用户登陆到集群。

- 您在网络策略所在的命名空间中。

流程

- 要删除网络策略对象，请输入以下命令：

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

其中：

<policy_name>

指定网络策略的名称。

<namespace>

可选：如果对象在与当前命名空间不同的命名空间中定义，使用它来指定命名空间。

输出示例

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

注意

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

8.3.5.2. 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。
- 您在 AWS 集群上创建了 Red Hat OpenShift Service。
- 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。

8.3.6. 为项目定义默认网络策略

作为集群管理员，您可以在创建新项目时修改新项目模板，使其自动包含网络策略。如果您还没有新项目的自定义模板，则需要首先创建一个。

8.3.6.1. 为新项目修改模板

作为集群管理员，您可以修改默认项目模板，以便使用自定义要求创建新项目。

创建自己的自定义项目模板：

前提条件

- 您可以使用具有 dedicated-admin 权限的账户访问 Red Hat OpenShift Service on AWS 集群。

流程

1. 以具有 cluster-admin 特权的用户身份登录。
2. 生成默认项目模板：

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 使用文本编辑器，通过添加对象或修改现有对象来修改生成的 template.yaml 文件。
4. 项目模板必须创建在 openshift-config 命名空间中。加载修改后的模板：

```
$ oc create -f template.yaml -n openshift-config
```

5. 使用 Web 控制台或 CLI 编辑项目配置资源。

- 使用 Web 控制台：

  i. 导航至 Administration → Cluster Settings 页面。

  ii. 单击 Configuration 以查看所有配置资源。

  iii. 找到 Project 的条目，并点击 Edit YAML。

- 使用 CLI：

  i. 编辑 project.config.openshift.io/cluster 资源：

```
$ oc edit project.config.openshift.io/cluster
```

6. 更新 spec 部分，使其包含 projectRequestTemplate 和 name 参数，再设置您上传的项目模板的名称。默认名称为 project-request。

带有自定义项目模板的项目配置资源

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
# ...
spec:
  projectRequestTemplate:
    name: <template_name>
# ...
```

7. 保存更改后，创建一个新项目来验证是否成功应用了您的更改。

8.3.6.2. 在新项目模板中添加网络策略

作为集群管理员，您可以在新项目的默认模板中添加网络策略。Red Hat OpenShift Service on AWS 将自动创建项目中模板中指定的所有 NetworkPolicy 对象。

前提条件

- 集群使用支持 NetworkPolicy 对象的默认 CNI 网络插件，如设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您需要使用具有 cluster-admin 权限的用户登陆到集群。
- 您必须已为新项目创建了自定义的默认项目模板。

流程

1. 运行以下命令来编辑新项目的默认模板：

```
$ oc edit template <project_template> -n openshift-config
```

将 <project_template> 替换为您为集群配置的缺省模板的名称。默认模板名称为 project-request。

2. 在模板中，将每个 NetworkPolicy 对象作为一个元素添加到 objects 参数中。objects 参数可以是一个或多个对象的集合。

在以下示例中，objects 参数集合包括几个 NetworkPolicy 对象。

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

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

3. 可选：通过运行以下命令创建一个新项目，来确认您的网络策略对象已被成功创建：

   a. 创建一个新项目：

   ```
   $ oc new-project <project> 1
   ```

   1 将 <project> 替换为您要创建的项目的名称。

   b. 确认新项目模板中的网络策略对象存在于新项目中：

   ```
   $ oc get networkpolicy
   NAME                          POD-SELECTOR   AGE
   allow-from-openshift-ingress   <none>        7s
   allow-from-same-namespace      <none>        7s
   ```

-

8.3.7. 使用网络策略配置多租户隔离

作为集群管理员，您可以配置网络策略以为多租户网络提供隔离功能。

注意

如果使用 OpenShift SDN 网络插件，请按照本节所述配置网络策略，提供类似于多租户模式的网络隔离，但设置了网络策略模式。

8.3.7.1. 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 网络插件或设置了 mode: NetworkPolicy 的 OpenShift SDN 网络插件。此模式是 OpenShift SDN 的默认模式。
- 已安装 OpenShift CLI（oc）。
- 您可以使用具有 admin 权限的用户登陆到集群。

流程

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

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

```
      policy-group.network.openshift.io/ingress: ""
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

注意

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

b.

名为 allow-from-openshift-monitoring 的策略：

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

c.

名为 allow-same-namespace 的策略：

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

d.

名为 allow-from-kube-apiserver-operator 的策略：

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

如需了解更多详细信息，请参阅 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
```

8.4. RED HAT OPENSHIFT SERVICE ON AWS 中的 INGRESS NODE FIREWALL OPERATOR

Ingress Node Firewall Operator 允许管理员在节点级别管理防火墙配置。

8.4.1. Ingress Node Firewall Operator

Ingress Node Firewall Operator 通过将守护进程集部署到您在防火墙配置中指定和管理的节点，在节点级别提供入口防火墙规则。要部署守护进程集，请创建一个 IngressNodeFirewallConfig 自定义资源 (CR)。Operator 应用 IngressNodeFirewallConfig CR 来创建入口节点防火墙守护进程集 daemon，它在与 nodeSelector 匹配的所有节点上运行。

您可以配置 IngressNodeFirewall CR 的规则，并使用 nodeSelector 将值设置为 "true" 的集群。

重要

Ingress Node Firewall Operator 仅支持无状态防火墙规则。

不支持原生 XDP 驱动程序的网络接口控制器 (NIC) 将以较低性能运行。

对于 Red Hat OpenShift Service on AWS 4.14 或更高版本，您必须在 RHEL 9.0 或更高版本上运行 Ingress Node Firewall Operator。

8.4.2. 安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 Red Hat OpenShift Service on AWS CLI 或 Web 控制台安装 Ingress Node Firewall Operator。

8.4.2.1. 使用 CLI 安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 CLI 安装 Operator。

前提条件

- 已安装 OpenShift CLI(oc)。
- 有管理员特权的帐户。

流程

1. 运行以下命令来创建 openshift-ingress-node-firewall 命名空间：

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
EOF
```

2. 运行以下命令来创建 OperatorGroup CR：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. 订阅 Ingress Node Firewall Operator。

   a. 要为 Ingress Node Firewall Operator 创建 Subscription CR，请输入以下命令：

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ingress-node-firewall-sub
```

```
  namespace: openshift-ingress-node-firewall
spec:
  name: ingress-node-firewall
  channel: stable
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. 要验证是否已安装 Operator，请输入以下命令：

```
$ oc get ip -n openshift-ingress-node-firewall
```

输出示例

```
NAME            CSV                                   APPROVAL    APPROVED
install-5cvnz   ingress-node-firewall.4.0-202211122336   Automatic   true
```

5. 要验证 Operator 的版本，请输入以下命令：

```
$ oc get csv -n openshift-ingress-node-firewall
```

输出示例

```
NAME                                         DISPLAY                       VERSION               REPLACES
PHASE
ingress-node-firewall.4.0-202211122336   Ingress Node Firewall Operator   4.0-
202211122336   ingress-node-firewall.4.0-202211102047   Succeeded
```

8.4.2.2. 使用 Web 控制台安装 Ingress Node Firewall Operator

作为集群管理员，您可以使用 Web 控制台安装 Operator。

前提条件

- 已安装 OpenShift CLI(oc)。
- 有管理员特权的帐户。

流程

1. 安装 Ingress Node Firewall Operator：
   a. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → OperatorHub。
   b. 从可用的 Operator 列表中选择 Ingress Node Firewall Operator，然后点 Install。
   c. 在 Install Operator 页面中，在 Installed Namespace 下选择 Operator recommended Namespace。
   d. 点 Install。
2. 验证 Ingress Node Firewall Operator 是否已成功安装：
   a. 导航到 Operators → Installed Operators 页面。
   b. 确保 openshift-ingress-node-firewall 项目中列出的 Ingress Node Firewall Operator 的 Status 为 InstallSucceeded。

注意

在安装过程中，Operator 可能会显示 Failed 状态。如果安装过程结束后有 InstallSucceeded 信息，您可以忽略这个 Failed 信息。

如果 Operator 没有 InstallSucceeded 状态，请按照以下步骤进行故障排除：

- 检查 Operator Subscriptions 和 Install Plans 选项卡中的 Status 项中是否有任何错误。

- 进入到 Workloads → Pods 页面，在 openshift-ingress-node-firewall 项目中检查 pod 的日志。

- 检查 YAML 文件的命名空间。如果缺少注解，您可以使用以下命令将注解 workload.openshift.io/allowed=management 添加到 Operator 命名空间中：

```
$ oc annotate ns/openshift-ingress-node-firewall
workload.openshift.io/allowed=management
```

注意

对于单节点 OpenShift 集群，openshift-ingress-node-firewall 命名空间需要 workload.openshift.io/allowed=management 注解。

8.4.3. 部署 Ingress Node Firewall Operator

前提条件

- 已安装 Ingress Node Firewall Operator。

流程

要拒绝 Ingress Node Firewall Operator，请创建一个 IngressNodeFirewallConfig 自定义资源，该资源将部署 Operator 的守护进程集。您可以通过应用防火墙规则，将一个或多个 IngressNodeFirewall CRD 部署到节点。

1. 在 openshift-ingress-node-firewall 命名空间中创建 IngressNodeFirewallConfig，名为 ingressnodefirewallconfig。

2. 运行以下命令来部署 Ingress Node Firewall Operator 规则：

```
$ oc apply -f rule.yaml
```

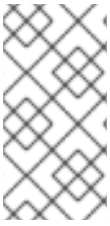
8.4.3.1. Ingress 节点防火墙配置对象

下表中描述了 Ingress Node Firewall 配置对象的字段：

表 8.1. Ingress 节点防火墙配置对象

字段	类型	描述
metadata.name	**string**	CR 对象的名称。防火墙规则对象的名称必须是 **ingressnodefirewallconfig**。
metadata.namespace	**string**	Ingress Firewall Operator CR 对象的命名空间。**IngressNodeFirewallConfig** CR 必须在 **openshift-ingress-node-firewall** 命名空间中创建。

spec.nodeSelector	**string**	通过指定节点标签 (label) 用于目标节点的节点选择约束。例如：
		`spec:`
		`  nodeSelector:`
		`  node-role.kubernetes.io/worker: ""`
		 
		注意
		nodeSelector 中使用的一个标签必须与节点上的标签匹配，以便守护进程集启动。例如，如果节点标签 **node-role.kubernetes.io/worker** 和 **node-type.kubernetes.io/vm** 应用到某个节点，则必须使用 **nodeSelector** 至少设置一个标签设置来使守护进程启动。



注意

Operator 使用 CR，并在与 nodeSelector 匹配的所有节点上创建一个入口节点防火墙守护进程集。

Ingress Node Firewall Operator 示例配置

以下示例中指定了完整的 Ingress Node 防火墙配置：

Ingress 节点防火墙配置对象示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
```

```
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

注意

Operator 使用 CR，并在与 nodeSelector 匹配的所有节点上创建一个入口节点防火墙守护进程集。

8.4.3.2. Ingress 节点防火墙规则对象

下表中描述了 Ingress Node Firewall 规则对象的字段：

表 8.2. Ingress 节点防火墙规则对象

字段	类型	描述
metadata.name	**string**	CR 对象的名称。
interfaces	**数组**	此对象的字段指定要应用防火墙规则的接口。例如，**- en0** 和 **- en1**。
nodeSelector	**数组**	您可以使用 **nodeSelector** 来选择节点来应用防火墙规则。将指定 **nodeselector** 标签的值设置为 **true** 以应用该规则。
ingress	**object**	**Ingress** 允许您配置允许从外部访问集群中的服务的规则。

Ingress 对象配置

ingress 对象的值在下表中定义：

表 8.3. Ingress 对象

字段	类型	描述

字段	类型	描述
sourceCIDRs	**数组**	允许您设置 CIDR 块。您可以从不同地址系列配置多个 CIDR。 **注意** 不同的 CIDR 允许您使用相同的顺序规则。如果同一节点有多个 **IngressNodeFirewall** 对象，且带有重叠 CIDR 的接口，则 **order** 字段将指定首先应用该规则。规则以升序应用。
rules	**数组**	对于每个 **source.CIDR**，Ingress 防火墙 **rules.order** 对象的顺序以 **1** 开始，每个 CIDR 最多 100 个规则。低顺序规则会首先执行。 **rules.protocolConfig.protocol** 支持以下协议：TCP、UDP、SCTP、ICMP 和 ICMPv6。ICMP 和 ICMPv6 规则可以匹配 ICMP 和 ICMPv6 类型或代码。TCP、UDP 和 SCTP 规则针对单一一个目标端口或一个目标端口范围（格式为 **<start : end-1>**）进行匹配。 将 **rules.action** 设置为 **allow** 以应用规则，**deny** 来禁止规则。 **注意** Ingress 防火墙规则使用阻止任何无效配置的验证 Webhook 进行验证。验证 Webhook 会阻止阻塞任何关键集群服务，如 API 服务器。

Ingress 节点防火墙规则对象示例

以下示例中指定了完整的 Ingress Node 防火墙配置：

Ingress 节点防火墙配置示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
```

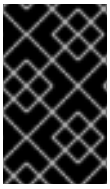
```
  name: ingressnodefirewall
spec:
  interfaces:
  - eth0
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> (1)
  ingress:
  - sourceCIDRs:
      - 172.16.0.0/12
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMP
        icmp:
          icmpType: 8 #ICMP Echo request
      action: Deny
    - order: 20
      protocolConfig:
        protocol: TCP
        tcp:
          ports: "8000-9000"
      action: Deny
  - sourceCIDRs:
      - fc00:f853:ccd:e793::0/64
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMPv6
        icmpv6:
          icmpType: 128 #ICMPV6 Echo request
      action: Deny
```

(1) 节点上必须存在 <label_name> 和 <label_value>，且必须与应用到您希望 ingressfirewallconfig CR 运行的节点的 nodeselector 标签和值匹配。<label_value> 可以是 true 或 false。通过使用 nodeSelector 标签，您可以针对单独的节点组为目标，以使用 ingressfirewallconfig CR 应用不同的规则。

零信任 Ingress Node Firewall 规则对象示例

零信任 Ingress 节点防火墙规则可为多接口集群提供额外的安全性。例如，您可以使用零信任 Ingress Node Firewall 规则来丢弃除 SSH 之外的特定接口上的网络流量。

以下示例中指定了零信任 Ingress Node Firewall 规则集的完整配置：



重要

用户需要为其提供应用程序使用的所有端口添加到允许列表，以确保正常工作。

零信任 Ingress 节点防火墙规则示例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
 name: ingressnodefirewall-zero-trust
spec:
 interfaces:
 - eth1 1
 nodeSelector:
   matchLabels:
     <ingress_firewall_label_name>: <label_value> 2
 ingress:
 - sourceCIDRs:
     - 0.0.0.0/0 3
   rules:
   - order: 10
     protocolConfig:
       protocol: TCP
       tcp:
         ports: 22
     action: Allow
   - order: 20
     action: Deny 4
```

1

Network-interface 集群

2

<label_name> 和 <label_value> 需要与应用到 ingressfirewallconfig CR 的特定节点的 nodeSelector 标签和值匹配。

3

0.0.0.0/0 匹配任何 CIDR

4

8.4.4. 查看 Ingress Node Firewall Operator 规则

流程

1. 运行以下命令来查看所有当前规则：

```
$ oc get ingressnodefirewall
```

2. 选择返回的 <resource> 名称之一，并运行以下命令来查看规则或配置：

```
$ oc get <resource> <name> -o yaml
```

8.4.5. 对 Ingress Node Firewall Operator 进行故障排除

- 运行以下命令列出已安装的 Ingress Node Firewall 自定义资源定义 (CRD)：

```
$ oc get crds | grep ingressnodefirewall
```

输出示例

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
ingressnodefirewallconfigs.ingressnodefirewall.openshift.io      2022-08-25T10:03:01Z
ingressnodefirewallnodestates.ingressnodefirewall.openshift.io   2022-08-
25T10:03:00Z
ingressnodefirewalls.ingressnodefirewall.openshift.io            2022-08-25T10:03:00Z
```

- 运行以下命令，以查看 Ingress Node Firewall Operator 的状态：

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

输出示例

```
NAME                                   READY  STATUS    RESTARTS  AGE
ingress-node-firewall-controller-manager  2/2   Running   0         5d21h
ingress-node-firewall-daemon-pqx56        3/3   Running   0         5d21h
```

以下字段提供有关 Operator 状态的信息： READY、STATUS、AGE、和 RESTARTS。当 Ingress Node Firewall Operator 将守护进程集部署到分配的节点时，STATUS 字段为 Running。

- 运行以下命令来收集所有入口防火墙节点 pod 的日志：

```
$ oc adm must-gather – gather_ingress_node_firewall
```

在 sos 节点的报告中，其中包含位于 /sos_commands/ebpf 的 eBPF bpftool 输出的报告。这些报告包括用于或作为入口防火墙 XDP 处理数据包处理、更新统计信息和发出事件的查找表。

第 9 章 OVN-KUBERNETES 网络插件

9.1. 配置出口 IP 地址

作为集群管理员，您可以配置 OVN-Kubernetes Container Network Interface (CNI) 网络插件，为命名空间分配一个或多个出口 IP 地址，或分配给命名空间中的特定 pod。

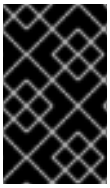
9.1.1. 出口 IP 地址架构设计和实施

Red Hat OpenShift Service on AWS 出口 IP 地址功能允许您确保来自一个或多个命名空间中的一个或多个 pod 的流量为集群网络外的服务具有一致的源 IP 地址。

例如，您可能有一个 pod 定期查询托管在集群外服务器上的数据库。要强制对服务器进行访问要求，将数据包过滤设备配置为只允许来自特定 IP 地址的流量。为确保您可以可靠地允许从该特定 pod 访问服务器，您可以为向服务器发出请求的 pod 配置特定的出口 IP 地址。

分配给命名空间的出口 IP 地址与用来向特定目的地发送流量的出口路由器不同。

在带有 HCP 集群的 ROSA 中，应用程序 pod 和入口路由器 pod 在同一节点上运行。如果您在这种情况下为应用程序项目配置出口 IP 地址，当您向应用程序项目发送请求时，不会使用 IP 地址。



重要

不支持使用 EgressIP 功能将出口 IP 地址分配给 control plane 节点。

以下示例演示了来自多个公共云提供商上节点的注解。注解被缩进以便于阅读。

AWS 上的 cloud.network.openshift.io/egress-ipconfig 注解示例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

以下小节描述了支持公共云环境的 IP 地址容量，用于容量计算。

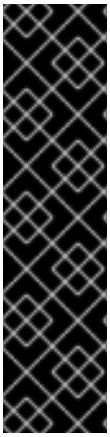
9.1.1.1. Amazon Web Services(AWS)IP 地址容量限制

在 AWS 上，IP 地址分配的限制取决于配置的实例类型。如需更多信息，请参阅 每个实例类型的每个网络接口的 IP 地址

9.1.1.2. 将出口 IP 分配给 pod

要将一个或多个出口 IP 分配给命名空间中的命名空间或特定 pod,必须满足以下条件：

- 集群中至少有一个节点必须具有 k8s.ovn.org/egress-assignable: "" 标签。
- 存在一个 EgressIP 对象定义一个或多个出口 IP 地址，用作从命名空间中离开集群的流量的源 IP 地址。



重要

如果您在为出口 IP 分配标记集群中的任何节点前创建 EgressIP 对象，Red Hat OpenShift Service on AWS 可能会将每个出口 IP 地址分配给第一个带有 k8s.ovn.org/egress-assignable: "" 标签的节点。

要确保出口 IP 地址在集群中的不同节点广泛分发，请在创建任何 EgressIP 对象前，始终将标签应用到您想托管出口 IP 地址的节点。

9.1.1.3. 将出口 IP 分配给节点

在创建 EgressIP 对象时，以下条件适用于标记为 k8s.ovn.org/egress-assignable: "" 标签的节点：

- 每次不会将出口 IP 地址分配给多个节点。

- 出口 IP 地址可在可以托管出口 IP 地址的可用节点之间平衡。
- 如果 EgressIP 对象中的 spec.EgressIPs 数组指定了多个 IP 地址，则适用以下条件：
  - 任何节点都不会托管超过一个指定的 IP 地址。
  - 流量在给定命名空间的指定 IP 地址之间大致相等。
- 如果节点不可用，则会自动重新分配给它的所有出口 IP 地址，但符合前面描述的条件。

当 Pod 与多个 EgressIP 对象的选择器匹配时，无法保证在 EgressIP 对象中指定的出口 IP 地址被分配为 pod 的出口 IP 地址。

另外，如果 EgressIP 对象指定了多个出口 IP 地址，则无法保证可以使用哪些出口 IP 地址。例如，如果 pod 与带有两个出口 IP 地址 (10.10.20.1 和 10.10.20.2) 的 EgressIP 对象的选择器匹配，其中任何一个都可以用于每个 TCP 连接或 UDP 对话。

9.1.1.4. 出口 IP 地址配置架构图

下图显示了出口 IP 地址配置。图中描述了，在一个集群的三个节点上运行的两个不同命名空间中的四个 pod。节点从主机网络上的 192.168.126.0/18 CIDR 块中分配 IP 地址。

Node 1 和 Node 3 都标记为 k8s.ovn.org/egress-assignable: ""，因此可用于分配出口 IP 地址。

图中的横线描述了 pod1、pod2 和 pod 3 的流量流，通过 pod 网络来从 Node 1 和 Node 3 出口集群。当外部服务从示例 EgressIP 对象选择的任何 pod 接收流量时，源 IP 地址为 192.168.126.10 或

192.168.126.102。这两个节点之间流量大致平衡。

图中的以下资源被详细描述：

命名空间对象

命名空间在以下清单中定义：

命名空间对象

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1
kind: Namespace
metadata:
  name: namespace2
  labels:
    env: prod
```

EgressIP 对象

以下 EgressIP 对象描述了一个配置，该配置选择将 env 标签设置为 prod 的任意命名空间中的所有 pod。所选 pod 的出口 IP 地址为 192.168.126.10 和 192.168.126.102。

EgressIP 对象

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egressips-prod
spec:
  egressIPs:
  - 192.168.126.10
  - 192.168.126.102
  namespaceSelector:
```

```
    matchLabels:
      env: prod
  status:
    items:
    - node: node1
      egressIP: 192.168.126.10
    - node: node3
      egressIP: 192.168.126.102
```

对于上例中的配置，Red Hat OpenShift Service on AWS 为可用的节点分配两个出口 IP 地址。status 字段显示是否以及在哪里分配了出口 IP 地址。

9.1.2. EgressIP 对象

以下 YAML 描述了 EgressIP 对象的 API。对象有效的范围为集群，它不是在命名空间中创建的。

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: <name> 1
spec:
  egressIPs: 2
  - <ip_address>
  namespaceSelector: 3
    ...
  podSelector: 4
    ...
```

1

EgressIPs 对象的名称。

2

包括一个或多个 IP 地址的数组。

3

出口 IP 地址与其关联的一个或多个命名空间选择器将。

4

以下 YAML 描述了命名空间选择器的小节：

命名空间选择器小节

```
namespaceSelector: 1
  matchLabels:
    <label_name>: <label_value>
```

1

命名空间的一个或多个匹配规则。如果提供多个匹配规则，则会选择所有匹配的命名空间。

以下 YAML 描述了 pod 选择器的可选小节：

Pod 选择器片段

```
podSelector: 1
  matchLabels:
    <label_name>: <label_value>
```

1

可选：与指定 namespaceSelector 规则匹配的命名空间中 pod 的一个或多个匹配规则。如果指定，则仅选择匹配的 pod。命名空间中的其他 Pod 不会被选择。

在以下示例中，EgressIP 对象将 192.168.126.11 和 192.168.126.102 出口 IP 地址与将 app 标签设置为 web 的 pod 关联，并位于将 env 标签设置为 prod 的命名空间中：

EgressIP 对象示例

-

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group1
spec:
  egressIPs:
  - 192.168.126.11
  - 192.168.126.102
  podSelector:
    matchLabels:
      app: web
  namespaceSelector:
    matchLabels:
      env: prod
```

在以下示例中，EgressIP 对象将 192.168.127.30 和 192.168.127.40 出口 IP 地址与任何没有将 environment 标签设置为 development 的 pod 相关联：

EgressIP 对象示例

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group2
spec:
  egressIPs:
  - 192.168.127.30
  - 192.168.127.40
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: NotIn
      values:
      - development
```

9.1.3. 标记节点以托管出口 IP 地址

您可以将 k8s.ovn.org/egress-assignable="" 标签应用到集群中的节点，以便 Red Hat OpenShift Service on AWS 可以为节点分配一个或多个出口 IP 地址。

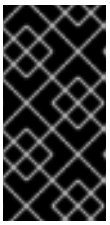
前提条件

- 安装 ROSA CLI (rosa)。
- 以集群管理员身份登录集群。

流程

- 要标记节点，使其可以托管一个或多个出口 IP 地址，请输入以下命令：

```
$ rosa edit machinepool <machinepool_name> --cluster=<cluster_name> --labels
"k8s.ovn.org/egress-assignable="
```



重要

此命令替换了 machinepool 中的任何激动的节点标签。您应该将任何所需的标签包含到 --labels 字段，以确保您的现有节点标签持久存在。

9.1.4. 后续步骤

- 分配出口 IP

9.1.5. 其他资源

- labelSelector meta/v1
- LabelSelectorRequirement meta/v1

第 10 章 配置路由

10.1. 路由配置

10.1.1. 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}
```

10.1.2. 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
```

10.1.3. 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。

10.1.3.1. 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）。

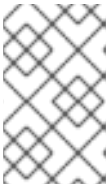
流程

- 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。

10.1.3.2. 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}}'
```

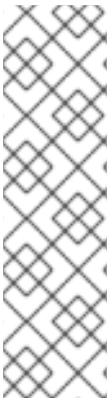
输出示例

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

10.1.4. 使用 Cookie 来保持路由有状态性

Red Hat OpenShift Service on AWS 提供粘性会话，通过确保所有流量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。

Red Hat OpenShift Service on AWS 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。

10.1.4.1. 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
```

10.1.5. 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 10.1. 路由可用性

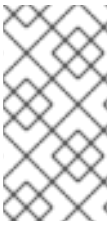
Route（路由）	当比较到	可访问
www.example.com/test	*www.example.com/test*	是
	www.example.com	否
www.example.com/test 和 *www.example.com*	*www.example.com/test*	是
	www.example.com	是
www.example.com	*www.example.com/text*	yes（由主机匹配，而不是路由）
	www.example.com	是

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。

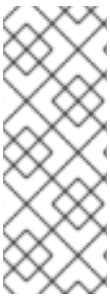


注意

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。

10.1.6. HTTP 标头配置

Red Hat OpenShift Service on AWS 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。

10.1.6.1. 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。

10.1.6.2. 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 10.2. 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

10.1.7. 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。

前提条件

- 已安装 OpenShift CLI(oc)。
- 以项目管理员身份登录到 Red Hat OpenShift Service on AWS 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*。

10.1.8. 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 10.3. 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**。默认值为 TLS passthrough 路由 **的源**。对于所有其他路由，默认值是 **随机的**。	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。allowlist 是以空格分隔的 IP 地址和 CIDR 范围列表，用于批准的源地址。来自允许列表中的 IP 地址的请求会被丢弃。 **haproxy.config** 文件中直接看到的最大 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，它们将写入从 haproxy.config 引用的独立文件中。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 对象大小限制路由注解的大小限制。因此，它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。

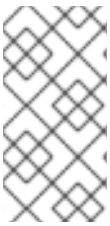
变量	默认	Description
ROUTER_CLIENT_FIN_TIMEOUT	**1s**	控制连接到路由的客户端的 TCP FIN 超时周期。如果发送到关闭连接的 FIN 在给定时间内没有回答，HAProxy 会关闭连接。如果设置为较低值，并且在路由器上使用较少的资源，则这不会产生任何损害。
ROUTER_DEFAULT_CLIENT_TIMEOUT	**30s**	客户端必须确认或发送数据的时长。
ROUTER_DEFAULT_CONNECT_TIMEOUT	**5s**	最长连接时间。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	**1s**	控制路由器到支持路由的 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 10.4. 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 10.5. 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。

10.1.9. 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，则 Red Hat OpenShift Service on AWS 会生成不安全的路由。要创建使用默认入口证书生成安全边缘终止路由的 Ingress 对象，您可以指定一个空的 TLS 配置，如下所示：

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 Red Hat OpenShift Service on AWS 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。

10.1.10. 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

10.2. 安全路由

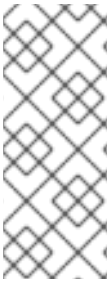
安全路由提供以下几种 TLS 终止功能来为客户端提供证书。以下小节介绍了如何使用自定义证书创建重新加密、边缘和透传路由。

10.2.1. 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-----
```

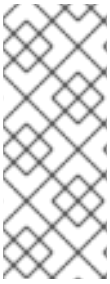
如需了解更多选项，请参阅 **oc create route reencrypt --help**。

10.2.2. 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。

10.2.3. 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

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。

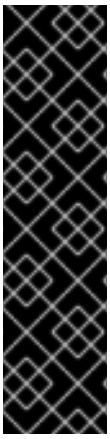
2 termination 字段设置为 passthrough。这是唯一需要 tls 的字段。

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。

10.2.4. 使用外部受管证书创建路由

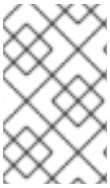


重要

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，通过第三方证书管理解决方案配置 Red Hat OpenShift Service on AWS 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进行故障排除，请检查 Red Hat OpenShift Service on AWS 路由器 pod 日志中的错误，请参阅 调查 pod 问题。