OpenShift Container Platform 4.18

网络概述

了解 OpenShift Container Platform 中的基本网络概念和常规任务

Last Updated: 2025-12-08

OpenShift Container Platform 4.18 网络概述

了解 OpenShift Container Platform 中的基本网络概念和常规任务

Legal Notice

Abstract

本文档介绍了 OpenShift Container Platform 中核心网络概念、基本架构和一般网络任务。

Table of Contents

第 1 章 了解网络

了解 OpenShift Container Platform 中网络的基本知识可确保集群中高效和安全通信，对于有效的网络管理至关重要。您环境中的网络关键元素包括了解 pod 和服务如何通信、IP 地址的角色以及使用 DNS 进行服务发现。

1.1. OPENSHIFT CONTAINER PLATFORM 中的网络

OpenShift Container Platform 可确保集群中不同组件之间以及外部客户端和集群间的无缝通信。网络依赖于以下核心概念和组件：

- pod 到 pod 的通信
- 服务
- DNS
- 入口
- 网络控制
- 负载平衡

1.1.1. 网络服务的常见实践

在 OpenShift Container Platform 中，服务会为客户端创建一个 IP 地址，即使多个 pod 都提供该服务。此抽象支持无缝扩展、容错和滚动升级，而不会影响客户端。

网络安全策略管理集群中的流量。网络控制允许命名空间管理员为其 pod 定义入口和出口规则。通过使用网络管理策略，集群管理员可以建立命名空间策略，覆盖命名空间策略，或者在没有定义时设置默认策略。

出口防火墙配置控制 pod 的出站流量。这些配置设置可确保仅进行授权的通信。入口节点防火墙通过控制传入流量来保护节点。另外，Universal Data Network 管理集群中的数据流量。

1.1.2. 网络功能

OpenShift Container Platform 提供多个网络功能和增强。这些功能及改进如下：

- Ingress Operator 和 Route API：OpenShift Container Platform 包含实现 Ingress Controller API 的 Ingress Operator。此组件通过部署和管理支持高级路由配置和负载平衡的基于 HAProxy 的 Ingress Controller，实现对集群服务的外部访问。OpenShift Container Platform 使用 Route API 将上游 Ingress 对象转换为路由对象。路由特定于 OpenShift Container Platform 中的网络，但也可以使用第三方 Ingress Controller。
- 增强安全性：OpenShift Container Platform 提供高级网络安全功能，如出口防火墙和入口节点防火墙。
    - 出口防火墙 ：出口防火墙控制并限制集群中 pod 的出站流量。您可以设置规则来限制哪些外部主机或 pod 可与之通信的 IP 范围。
    - Ingress 节点防火墙：入口节点防火墙由 Ingress Firewall Operator 管理，并在节点级别提供防火墙规则。您可以通过在集群内的特定节点上配置此防火墙，以便在到达这些节点前过滤传入的流量来保护节点免受威胁。

> **注意**
>
> OpenShift Container Platform 还实施服务，如 Network Policy、Admin Network Policy 和 Security Context Constraints (SCC)，以保护 pod 之间的通信并强制实施访问控制。

- 基于角色的访问控制 (RBAC)：OpenShift Container Platform 会扩展 Kubernetes RBAC，以更精细地控制谁可以访问和管理网络资源。RBAC 有助于在集群中保持安全和合规性。
- 多租户支持：OpenShift Container Platform 提供多租户支持，使多个用户和团队能够共享同一集群，同时保持其资源隔离和安全。
- 混合云和多云功能：OpenShift Container Platform 旨在在内部、云和多云环境间无缝工作。这种灵活性使机构能够在不同的基础架构上部署和管理容器化应用。
- 可观察性和监控 ：OpenShift Container Platform 提供集成的可观察性和监控工具，可帮助管理并排除网络问题。这些工具包括对网络指标和日志的基于角色的访问。
- 用户定义的网络 (UDN)：UDN 允许管理员自定义网络配置。UDN 提供增强的网络隔离和 IP 地址管理。
- Egress IP： Egress IP 允许您为来自命名空间中的 pod 的所有出口流量分配固定源 IP 地址。出口 IP 可通过确保外部服务提供一致的源 IP 地址来提高安全性和访问控制。例如，如果 pod 需要访问只允许来自特定 IP 地址流量的外部数据库，您可以为该 pod 配置出口 IP 来满足访问要求。
- Egress 路由器：出口（Egress）路由器是一个 pod，充当集群和外部系统间的桥接。出口路由器允许来自 pod 的流量通过不用于任何其他目的的特定 IP 地址路由。使用出口路由器，您可以通过特定网关强制访问控制或路由流量。

1.2. 使用节点、客户端和集群联网

节点是集群中的一个机器，它可以运行 control-plane 组件、工作负载组件或两者。节点是一个物理服务器或一个虚拟机。集群是运行容器化应用程序的节点集合。客户端是与集群交互的工具和用户。

1.2.1. 什么是节点？

节点是运行容器化应用程序的物理或虚拟机。节点托管 pod 并提供资源，如用于运行应用的内存和存储。节点可以启用 pod 之间的通信。每个 pod 都分配一个 IP 地址。同一节点上的 Pod 可以使用这些 IP 地址相互通信。节点通过允许 pod 发现并与集群中的服务进行通信来协助服务发现。节点有助于在 pod 之间分发网络流量，以确保应用程序的有效负载平衡和高可用性。节点在内部集群网络和外部网络之间提供一个桥接，以允许外部客户端访问集群中运行的服务。

1.2.2. 了解集群

集群是一起运行容器化应用程序的节点集合。这些节点包括 control plane 节点和计算节点。

1.2.3. 了解外部客户端

外部客户端是集群外部的任何实体，与集群中运行的服务和应用程序交互。外部可以包含最终用户、外部服务和外部设备。最终用户是通过其浏览器或移动设备访问集群中托管的 Web 应用程序的人员。外部服务是与其他软件系统或与集群中服务交互的应用程序，通常使用 API。外部设备是需要与集群服务通信的集群网络之外的任何硬件，如互联网(IoT)设备。

1.3. 网络概念和组件

OpenShift Container Platform 中的网络使用多个关键组件和概念。

- Pod 和服务是 Kubernetes 中的最小可部署单元，服务为一组 pod 提供稳定的 IP 地址和 DNS 名称。集群中的每个 pod 都会被分配一个唯一的 IP 地址。Pod 使用 IP 地址直接与其他 pod 通信，无论它们所处的节点是什么。当 pod 被销毁和创建时，pod IP 地址将改变。另外，还会为服务分配唯一的 IP 地址。服务与可以提供该服务的 pod 关联。在访问后，服务 IP 地址会提供一个稳定的方式来访问 pod，它向用于支持服务的一个 pod 发送网络数据。

- Route 和 Ingress API 定义将 HTTP、HTTPS 和 TLS 流量路由到集群内服务的规则。OpenShift Container Platform 提供 Route 和 Ingress API 作为默认安装的一部分，但您可以在集群中添加第三方 Ingress Controller。

- Container Network Interface (CNI) 插件管理 pod 网络，以启用 pod 到 pod 的通信。

- Cluster Network Operator (CNO) CNO 管理集群的网络插件组件。使用 CNO，您可以设置网络配置，如 pod 网络 CIDR 和服务网络 CIDR。

- DNS operator 管理集群中的 DNS 服务，以确保可通过 DNS 名称访问服务。

- 网络控制定义 pod 允许相互通信和其他网络端点。这些策略通过控制 Pod 通信的流量流和强制实施规则来帮助保护集群的安全。

- 负载平衡在多个服务器之间分发网络流量，以确保可靠性和性能。

- 服务发现是服务查找和在集群中相互通信的一种机制。

- Ingress Operator 使用 OpenShift Container Platform Route 来管理路由并启用对集群服务的外部访问。

其他资源

- 关于网络策略

1.4. POD 如何通信

Pod 使用 IP 地址进行通信，并使用 DNS 来发现 pod 或服务的 IP 地址。集群使用各种策略类型来控制允许的通信。Pod 以两种方式通信： pod 到 pod 的通信和服务到 pod 的通信。

1.4.1. pod 到 pod 的通信

pod 到 pod 的通信代表 pod 能够在集群中相互通信。这对微服务和分布式应用的功能至关重要。

集群中的每个 pod 都会被分配一个唯一 IP 地址，用来直接与其他 pod 通信。Pod 到 pod 的通信对于集群内部通信非常有用，Pod 需要相互间交换数据或协作执行任务。例如，Pod A 可以使用 Pod B 的 IP 地址直接向 Pod B 发送请求。pod 可以在没有网络地址转换(NAT)的情况下通过扁平网络通信。这允许在不同节点间的 pod 间无缝通信。

1.4.1.1. 示例：控制 pod 到 pod 的通信

在有多个容器的基于微服务的应用中，前端 pod 需要与后端 pod 通信来检索数据。通过使用 pod 到 pod 的通信（直接或通过服务进行），这些 pod 可以有效地交换信息。

要控制和保护 pod 到 pod 的通信，您可以定义网络控制。这些控制用来强制实施安全性和合规要求，它基于标签和选择器指定 pod 如何相互交流。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-some-pods
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: backend
      ports:
        - protocol: TCP
          port: 80
```

1.4.2. 服务到 pod 的通信

服务到 pod 通信可确保服务能够可靠地将网络流量路由到适当的 pod。服务是对象，用于定义 pod 的逻辑集合并提供稳定的端点，如 IP 地址和 DNS 名称。Pod IP 地址可能会改变。服务抽象 pod IP 地址，以提供一致的访问应用程序组件，即使 IP 地址有变化。

服务到 pod 通信的主要概念包括：

- 端点：端点定义了与服务关联的 pod 的 IP 地址和端口。
- 选择器：选择器（Selector）使用标签（如键值对）来定义选择服务应目标的一组对象的条件。
- 服务：服务为一组 pod 提供稳定的 IP 地址和 DNS 名称。此抽象允许其他组件与服务而不是单个 pod 通信。
- 服务发现：DNS 使服务可被发现。在创建服务时，会为其分配一个 DNS 名称。其他 pod 发现此 DNS 名称，并使用它来与服务通信。
- 服务类型 ：服务类型控制如何在集群内部或外部公开服务。
  - ClusterIP 在内部集群 IP 上公开服务。它是默认的服务类型，使该服务只能从集群内部访问。
  - NodePort 允许外部流量来访问服务，它使用节点的 IP 通过一个静态端口公开服务。
  - LoadBalancer 使用云供应商的负载均衡器向外部公开服务。

服务使用选择器来标识应接收流量的 pod。选择器与 pod 上的标签匹配，以确定哪些 pod 是服务的一部分。示例：带有选择器 **app: myapp** 的服务，会将流量路由到带有标签 **app: myapp** 的所有 pod。

端点会被动态更新，以反映与服务选择器匹配的 pod 的当前 IP 地址。{product-name} 维护这些端点，并确保服务将流量路由到正确的 pod。

通信流指的是 Kubernetes 中的服务将流量路由到适当 pod 时发生的步骤和交互序列。服务到 pod 通信的典型通信流如下：

- 服务创建：当您创建服务时，您可以定义服务类型、服务侦听的端口以及选择器标签。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

- DNS 解析：每个 pod 都有一个 DNS 名称，其他 pod 可以使用该名称与服务通信。例如，如果服务在 **my-app** 命名空间中的名称为 **my-service**，其 DNS 名称为 **my-service.my-app.svc.cluster.local**。
- 流量路由 ：当 pod 将请求发送到服务的 DNS 名称时，OpenShift Container Platform 会将名称解析为服务的 ClusterIP。然后，服务使用端点将流量路由到与其选择器匹配的其中一个 pod。
- 负载平衡：服务还提供基本的负载平衡。它们在所有与选择器匹配的 pod 之间分发传入的流量。这样可确保没有单个 pod 消耗太多的流量。

1.4.2.1. 示例：控制服务到 pod 的通信

集群正在运行基于微服务的应用，它有两个组件：前端和后端。前端需要与后端通信来获取数据。

流程

1. 创建后端服务。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. 配置后端 pod。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: backend-pod
  labels:
    app: backend
spec:
  containers:
    - name: backend-container
      image: my-backend-image
      ports:
```

```
- containerPort: 8080
```

3. 建立前端通信。
   前端 pod 现在可以使用 DNS 名称 **backend.default.svc.cluster.local** 与后端服务通信。该服务可确保流量路由到其中一个后端 Pod。

服务到 pod 通信会抽象管理 pod IP 的复杂性，并确保集群中的可靠和高效的通信。

1.5. 支持的负载均衡器

负载均衡在多个服务器之间分发传入的网络流量，以确保没有单个服务器负载过大，以维护集群的健康和效率。负载均衡器是执行负载平衡的设备。它们充当客户端和服务器之间的中介，以根据预定义的规则管理和直接流量。

OpenShift Container Platform 支持以下负载均衡器类型：

- Classic Load Balancer (CLB)
- Elastic Load Balancing (ELB)
- Network Load Balancer (NLB)
- Application Load Balancer (ALB)

ELB 是 AWS 路由器的默认负载均衡器类型。CLB 是自我管理环境的默认设置。NLB 是 Red Hat OpenShift Service on AWS (ROSA) 的默认设置。

重要

在应用程序前面使用 ALB，而不是在路由器前面。使用 ALB 需要 AWS Load Balancer Operator 附加组件。所有 Amazon Web Services (AWS) 区域或所有 OpenShift Container Platform 配置集都不支持这个 Operator。

1.5.1. 配置负载均衡器

您可以在集群安装过程中定义默认负载均衡器类型。安装后，您可以将入口控制器配置为使用特定的方式操作，这个方式没有包括在在集群安装中订阅的全局平台配置。

1.5.1.1. 定义默认负载均衡器类型

安装集群时，您可以指定要使用的负载均衡器的类型。您在集群安装中选择的负载均衡器类型应用到整个集群。

本例演示了如何为 AWS 上部署的集群定义默认负载均衡器类型。您可以在其他支持的平台中应用该流程。

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
platform:
 aws: 1
  lbType: classic 2
```

1. **platform** 键代表部署集群的平台。这个示例使用 **aws**。

2. **lbType** 键代表负载均衡器类型。本例使用 Classic Load Balancer（ **classic** ）。

1.5.1.2. 指定一个 Ingress Controller 的负载均衡器行为

安装集群后，您可以配置 Ingress Controller 来指定服务如何公开到外部网络，以便您可以更好地控制负载均衡器的设置和行为。

注意

更改 Ingress Controller 上的负载均衡器设置可能会覆盖您在安装时指定的负载均衡器设置。

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
endpointPublishingStrategy:
  loadBalancer: (1)
    dnsManagementPolicy: Managed
    providerParameters:
      aws:
        classicLoadBalancer: (2)
          connectionIdleTimeout: 0s
        type: Classic
      type: AWS
    scope: External
  type: LoadBalancerService
```

1. 'loadBalancer' 字段指定负载均衡器配置设置。

2. **classicLoadBalancer** 字段将负载均衡器设置为 **classic**，并包含特定于 AWS 上 CLB 的设置。

1.6. 域名系统(DNS)

域名系统(DNS)是一个分层命名系统，用于将用户友好的域名（如 www.example.com）转换为标识网络中计算机的 IP 地址。DNS 在服务发现和名称解析方面扮演重要角色。

OpenShift Container Platform 提供了一个内置的 DNS，以确保可通过 DNS 名称访问服务。这有助于维护稳定的通信，即使底层 IP 地址有变化。当您启动 pod 时，会自动创建服务名称、IP 地址和端口的环境变量，以便 pod 与其他服务通信。

1.6.1. 主要 DNS 术语

- CoreDNS：CoreDNS 是 DNS 服务器，为服务和 pod 提供名称解析。

- DNS 名称 ：服务根据其命名空间和名称分配 DNS 名称。例如，在 **default** 命名空间中名为 **my-service** 的服务将具有 DNS 名称 **my-service.default.svc.cluster.local**。

- 域名：域名是用于访问网站和服务（如 **example.com** ）的用户友好名称。

- IP 地址：IP 地址是分配给连接到使用 IP 进行通信的计算机网络的每个设备的数字标签。IPv4 地址的示例是 **192.0.2.1**。一个 IPv6 地址的示例为 **2001:0db8:85a3:0000:0000:8a2e:0370:7334**。
- DNS 服务器：DNS 服务器是存储 DNS 记录的专用服务器。这些记录将域名映射到 IP 地址。当您在浏览器中输入域名时，您的计算机会联系 DNS 服务器来查找对应的 IP 地址。
- 解析过程：将 DNS 查询发送到 DNS 解析器。DNS 解析器随后联系一系列 DNS 服务器，以查找与域名关联的 IP 地址。解析器将尝试使用带有一系列域的名称，如 **<namespace>.svc.cluster.local**, **svc.cluster.local**, 和 **cluster.local**。这个过程会在第一个匹配项中停止。IP 地址返回到浏览器，然后使用 IP 地址连接到 Web 服务器。

1.6.2. 示例：DNS 用例

在本例中，前端应用在一组 pod 中运行，后端服务在另一组 pod 中运行。前端应用需要与后端服务通信。您可以为后端 pod 创建服务，为其提供稳定的 IP 地址和 DNS 名称。前端 pod 使用此 DNS 名称来访问后端服务，而不考虑各个 pod IP 地址的更改。

通过为后端 pod 创建服务，您可以提供一个稳定的 IP 和 DNS 名称 **backend-service.default.svc.cluster.local**，前端 pod 可用于与后端服务通信。此设置可确保即使单个 pod IP 地址有变化，通信仍保持一致且可靠。

以下步骤演示了如何配置前端 pod 以使用 DNS 与后端服务通信。

1. 创建后端服务。

   a. 部署后端 pod。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-container
          image: your-backend-image
          ports:
            - containerPort: 8080
```

   b. 定义服务以公开后端容器集。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
```

```
selector:
  app: backend
ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

2. 创建前端 pod。

   a. 定义前端 pod。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  labels:
    app: frontend
  spec:
    replicas: 3
    selector:
    matchLabels:
      app: frontend
   template:
   metadata:
     labels:
       app: frontend
   spec:
     containers:
       - name: frontend-container
         image: your-frontend-image
         ports:
           - containerPort: 80
```

   b. 将 pod 定义应用到集群。

```
$ oc apply -f frontend-deployment.yaml
```

3. 配置前端以与后端通信。
   在您的前端应用程序代码中，使用后端服务的 DNS 名称来发送请求。例如，如果您的前端应用程序需要从后端 pod 获取数据，则您的应用程序可能会包括以下代码：

```
fetch('http://backend-service.default.svc.cluster.local/api/data')
  .then(response => response.json())
  .then(data => console.log(data));
```

1.7. 网络控制

网络控制定义了 pod 允许相互通信和其他网络端点的规则。网络控制在网络级别实施，以确保只有允许的流量可以在 pod 之间流动。这有助于通过限制流量流并防止未经授权的访问来保护集群。

- 管理网络策略(ANP)：NP 是集群范围的自定义资源定义(CRD)。作为集群管理员，您可以使用 ANP 在集群级别上定义网络策略。您不能使用常规网络策略对象覆盖这些策略。这些策略在整个集群中强制实施严格的网络安全规则。ANPs 可以指定入口和出口规则，以便管理员控制进入和离开集群的流量。

- 出口防火墙 ：出口防火墙限制离开集群的出口流量。使用这个防火墙，管理员可以限制 Pod 可以从集群内部访问的外部主机。您可以配置出口防火墙策略，以允许或拒绝到特定 IP 范围、DNS 名称或外部服务的流量。这有助于防止未授权访问外部资源，并确保只有允许的流量可以离开集群。

- Ingress 节点防火墙：入口节点防火墙控制集群中节点的入口流量。使用这个防火墙，管理员定义限制哪些外部主机可以启动到节点的连接的规则。这有助于保护节点不受未授权访问的影响，并确保只有可信流量可以访问集群。

1.8. ROUTE 和 INGRESS

路由和入口都用于将应用程序公开给外部流量。但是，它们提供的目的略有不同，具有不同的功能。

1.8.1. Routes

路由特定于 OpenShift Container Platform 资源，它们通过主机名公开服务，以便外部客户端可根据名称访问服务。

路由将主机名映射到服务。路由名称映射允许使用主机名访问服务。路由为定向到该服务的流量提供负载均衡。路由中使用的主机名解析为路由器的 IP 地址。路由然后将流量转发到适当的服务。路由也可以使用 SSL/TLS 保护，以加密客户端和服务之间的流量。

1.8.2. 入口

Ingress 是一个提供高级路由功能的资源，包括负载均衡、SSL/TLS 终止和基于名称的虚拟主机。以下是有关 Ingress 的一些关键点：

- HTTP/HTTPS 路由 ：您可以使用 Ingress 定义将 HTTP 和 HTTPS 流量路由到集群中的服务的规则。

- Load balancing(负载均衡) ：Ingress Controller （如 NGINX 或 HAProxy）根据用户定义的规则管理流量路由和负载均衡。

- SSL/TLS termination(SSL/TLS 终止)：SSL/TLS 终止是在将传入的 SSL/TLS 流量传递给后端服务前解密传入的 SSL/TLS 流量的过程。

- 多个域和路径： Ingress 支持多个域和路径的路由流量。

1.8.3. 路由和入口的比较

与入口相比，路由提供了更大的灵活性和高级功能。这使得路由适合复杂路由场景。路由更易于设置和使用，特别是对于基本的外部访问需求。Ingress 通常用于简单直接的外部访问。路由用于更复杂的场景，需要高级路由和 SSL/TLS 终止。

1.8.4. 示例：配置路由和入口以公开 Web 应用程序

Web 应用程序在 OpenShift Container Platform 集群上运行。您希望外部用户访问应用程序。该应用应该可通过特定的域名访问，流量应该使用 TLS 安全加密。以下示例演示了如何配置路由和入口，以安全地将 Web 应用程序公开给外部流量。

1.8.4.1. 配置路由

1. 创建一个新项目。

```
$ oc new-project webapp-project
```

2. 部署 Web 应用。

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git --name=webapp
```

3. 使用路由公开服务。

```
$ oc expose svc/webapp --hostname=webapp.example.com
```

4. 使用 TLS 保护路由。
   a. 使用您的证书和密钥创建 TLS secret。

   ```
   $ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key
   ```

   b. 更新路由以使用 TLS secret。

   ```
   $ oc patch route/webapp -p '{"spec":{"tls":
   {"termination":"edge","certificate":"path/to/tls.crt","key":"path/to/tls.key"}}}'
   ```

1.8.4.2. 配置入口

1. 创建入口资源。
   确保在集群中安装并运行 ingress Controller。

2. 为 Web 应用创建服务。如果尚未创建，以服务的方式公开应用。

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
  namespace: webapp-project
spec:
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

3. 创建入口资源。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  namespace: webapp-project
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
    - host: webapp.example.com
```

```
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: webapp-service
                port:
                  number: 80
```

4. 使用 TLS 保护入口。

   a. 使用您的证书和密钥创建 TLS secret。

   ```
   $ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key -n webapp-project
   ```

   b. 更新入口资源以使用 TLS secret。

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: webapp-ingress
     namespace: webapp-project
   spec:
     tls: 1
       - hosts:
           - webapp.example.com
         secretName: webapp-tls 2
     rules:
       - host: webapp.example.com
         http:
           paths:
             - path: /
               pathType: Prefix
               backend:
                 service:
                   name: webapp-service
                   port:
                     number: 80
   ```

   1 **TLS** 部分指定 TLS 设置。

   2 **secretName** 字段是包含 TLS 证书和密钥的 Kubernetes secret 的名称。

1.9. 安全和流量管理

管理员可以使用服务类型将应用程序公开给外部流量和安全网络连接，如 **ClusterIP**、**NodePort** 和 **LoadBalaner** 和 API 资源，如 **Ingress** 和 **Route**。Ingress Operator 和 Cluster Network Operator (CNO) 可帮助配置和管理这些服务和资源。Ingress Operator 部署并管理一个或多个 Ingress Controller。这些控制器将外部 HTTP 和 HTTPS 流量路由到集群中的服务。CNO 部署和管理集群网络组件，包括 pod 网络、服务网络和 DNS。

1.9.1. 公开应用程序

ClusterIP 在集群中的内部 IP 上公开服务，以便集群只能被集群中的其他服务访问。NodePort 服务类型在每个节点的 IP 的静态端口上公开服务。此服务类型允许外部流量访问该服务。负载均衡器通常用于使用 MetalLB 的云或裸机环境中。此服务类型置备将外部流量路由到服务的外部负载均衡器。在裸机环境中，MetalLB 使用 VIP 和 ARP 公告或 BGP 公告。

Ingress 是一个 API 对象，用于管理对服务的外部访问，如负载均衡、SSL/TLS 终止和基于名称的虚拟主机。Ingress Controller （如 NGINX 或 HAProxy）实现 Ingress API，并根据用户定义的规则处理流量路由。

1.9.2. 保护连接

Ingress Controller 管理 SSL/TLS 终止，以便在将传入的 SSL/TLS 流量传递给后端服务前解密传入的 SSL/TLS 流量。SSL/TLS 终止从应用程序 pod 卸载加密/解密过程。您可以使用 TLS 证书加密客户端和服务之间的流量。您可以使用 **cert-manager** 等工具管理证书，以自动执行证书分发和续订。

如果存在 SNI 字段，将 TLS 流量传递给 pod。这个过程允许使用 TLS 而不是 HTTP/HTTPS 来公开运行 TCP 的服务。站点管理员可以集中管理证书，并允许应用程序开发人员在没有权限的情况下读取私钥。

Route API 启用使用集群管理的证书加密路由器到 pod 流量。这可确保集中管理外部证书，而内部部分仍加密。应用程序开发人员为其应用程序获得唯一的私钥。这些密钥可以作为 secret 挂载到 pod 中。

网络控制定义了 pod 如何相互通信的规则，以及其他网络端点。这通过控制集群内的流量流来增强安全性。这些控制在网络插件级别实现，以确保只允许 pod 间的流量流。

基于角色的访问控制(RBAC)管理权限并控制谁可以访问集群中的资源。服务帐户为访问 API 的 pod 提供身份。使用 RBAC 可以精细控制每个 pod 可以做什么。

1.9.3. 示例：公开应用程序并保护连接

在本例中，集群中运行的 Web 应用需要被外部用户访问。

1. 使用适合您的需要的服务类型，创建一个服务并将应用公开为服务。

```
apiVersion: v1
kind: Service
metadata:
  name: my-web-app
spec:
  type: LoadBalancer
  selector:
    app: my-web-app
  ports:
  - port: 80
    targetPort: 8080
```

2. 定义 **Ingress** 资源来管理 HTTP/HTTPS 流量并将其路由到您的服务。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
```

```
spec:
  rules:
  - host: mywebapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-web-app
            port:
              number: 80
```

3. 为您的入口配置 TLS，以确保安全、加密的连接。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
  - hosts:
    - mywebapp.example.com
    secretName: my-tls-secret
  rules:
  - host: mywebapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-web-app
            port:
              number: 80
```

1.9.4. 在服务类型和 API 资源之间进行选择

服务类型和 API 资源为公开应用和安全网络连接提供了不同的好处。通过利用适当的服务类型或 API 资源，您可以有效地管理应用程序公开方式，并确保对内部和外部客户端的安全、可靠的访问。

OpenShift Container Platform 支持以下服务类型和 API 资源：

- 服务类型
  - **ClusterIP** 用于仅限内部公开。易于设置，为访问集群中的服务提供稳定的内部 IP 地址。**ClusterIP** 适合集群中服务间的通信。
  - **NodePort** 允许通过在静态端口的每个节点 IP 上公开服务来实现外部访问。设置并可用于开发和测试非常简单。**NodePort** 非常适合简单的外部访问，无需来自云提供商的负载平衡器。
  - **LoadBalancer** 会自动置备外部负载均衡器来在多个节点之间分发流量。它是需要可靠、高可用性访问的生产环境的理想选择。

- **ExternalName** 将服务映射到外部 DNS 名称，以允许使用服务的 DNS 名称访问集群外的服务。最好将外部服务或旧系统与集群集成。
- 无头服务是一个 DNS 名称，可在不提供稳定的 **ClusterIP** 的情况下返回 pod IP 列表。这适用于需要直接访问各个 pod IP 的有状态应用程序或场景。

- API 资源
  - **Ingress** 提供对路由 HTTP 和 HTTPS 流量的控制，包括支持负载平衡、SSL/TLS 终止和基于名称的虚拟主机。它比仅服务更灵活，并支持多个域和路径。当需要复杂的路由时，**Ingress** 是理想的选择。
  - **Route** 与 **Ingress** 类似，但提供额外的功能，包括 TLS 重新加密和透传。它简化了向外部公开服务的过程。当需要高级功能（如集成证书管理）时，**Route** 时最佳选择。

如果您需要简单的方法来向外部流量公开服务，则 **Route** 或 **Ingress** 可能是最佳选择。这些资源可由命名空间 admin 或 developer 管理。最简单的方法是创建路由，检查其外部 DNS 名称，并将 DNS 配置为具有指向外部 DNS 名称的 CNAME。

对于 HTTP/HTTPS/TLS，**Route** 或 **Ingress** 应该会足够。如果使用其他方法会更加复杂，需要集群管理员来确保端口可以访问，或者配置了 MetalLB。在云环境中或适当配置的裸机环境中，**LoadBalancer** 服务也是一个选项。

第 2 章 访问主机

了解如何创建堡垒主机来访问 OpenShift Container Platform 实例，以及使用安全 shell (SSH) 访问 control plane 节点。

2.1. 访问安装程序置备的基础架构集群中 AMAZON WEB SERVICES 上的主机

OpenShift Container Platform 安装程序不会为任何置备 OpenShift Container Platform 集群的 Amazon Elastic Compute Cloud (Amazon EC2) 实例创建公共 IP 地址。为了可以 SSH 到 OpenShift Container Platform 主机，您必须按照以下步骤操作。

流程

1. 创建一个安全组，允许 SSH 访问由 **openshift-install** 命令创建的虚拟私有云 (VPC) 。
2. 在安装程序创建的某个公共子网中创建 Amazon EC2 实例。
3. 将公共 IP 地址与您创建的 Amazon EC2 实例相关联。
与 OpenShift Container Platform 安装不同，您应该将您创建的 Amazon EC2 实例与 SSH 密钥对关联。这与您为这个实例选择的操作系统无关，因为它只是一个 SSH 堡垒将互联网桥接到 OpenShift Container Platform 集群的 VPC。它与您使用的 Amazon Machine Image (AMI) 相关。例如，在 Red Hat Enterprise Linux CoreOS（RHCOS） 中，您可以像安装程序一样通过 Ignition 提供密钥。
4. 一旦置备了 Amazon EC2 实例并可以 SSH 到它，您必须添加与 OpenShift Container Platform 安装关联的 SSH 密钥。这个密钥可以与堡垒实例的密钥不同，也可以相同。

注意

直接通过 SSH 访问仅建议在灾难恢复时使用。当 Kubernetes API 正常工作时，应该使用特权 Pod。

5. 运行 **oc get nodes**，查看输出结果，然后选择一个 master 节点。主机名类似于 **ip-10-0-1-163.ec2.internal**。
6. 从您手动部署到 Amazon EC2 的堡垒 SSH 主机中，SSH 部署到该 control plane 主机。确定您使用了在安装过程中指定的相同的 SSH 密钥：

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

第 3 章 网络仪表板

网络指标可在 OpenShift Container Platform web 控制台的 dashboard 中的 **Observe → Dashboards** 下查看。

3.1. NETWORK OBSERVABILITY OPERATOR

如果安装了 Network Observability Operator，可以通过从 **Dashboards** 下拉列表中选择 **Netobserv** 仪表板来查看网络流量指标仪表板。有关此**仪表板**中可用指标的更多信息，请参阅 Network Observability metrics dashboard。

3.2. 网络和 OVN-KUBERNETES 仪表板

您可以从仪表板中查看常规网络指标和 OVN-Kubernetes 指标。

要查看常规网络指标，请从 **Dashboards** 下拉列表中选择 **Networking/Linux Subsystem Stats**。您可以从控制面板查看以下网络指标：**Network Utilisation**、**Network Saturation** 和 **Network Errors**。

要查看 OVN-Kubernetes 指标，请从 **Dashboards** 下拉列表中选择 **Networking/Infrastructure**。您可以查看以下 OVN-Kuberenetes 指标：**Networking Configuration**, **TCP Latency Probes**, **Control Plane Resources**, 和 **Worker Resources**。

3.3. INGRESS OPERATOR 仪表板

您可以从仪表板中查看 Ingress Operator 处理的网络指标。这包括类似如下的指标：

- 传入和传出带宽
- HTTP 错误率
- HTTP 服务器响应延迟

要查看这些 Ingress 指标，请从 **Dashboards** 下拉列表中选择 **Networking/Ingress**。您可以查看以下类别的 Ingress 指标： **Top 10 Per Route**, **Top 10 Per Namespace** 和 **Top 10 Per Shard**。

第 4 章 CIDR 范围定义

如果您的集群使用 OVN-Kubernetes，您必须为无类别域间路由(CIDR)子网范围指定非重叠范围。

重要

对于 OpenShift Container Platform 4.17 及更新的版本，集群使用 **169.254.0.0/17**（IPv4），或 **fd69::/112**（IPv6）作为默认的伪装子网。用户必须避免这些范围。对于升级的集群，默认 masquerade 子网没有变化。

以下子网类型，对于使用 OVN-Kubernetes 的集群是强制的：

- Join ：使用 join 交换将网关路由器连接到分布式路由器。join 交换减少了分布式路由器的 IP 地址数量。对于使用 OVN-Kubernetes 插件的集群，专用子网的 IP 地址都会被分配给附加到 join 交换的任何逻辑端口。
- Masquerade：在负载均衡器做出路由决策后，将从节点发送到同一节点的相同源和目标 IP 地址发生冲突。
- Transit ：传输开关是跨集群中所有节点的分布式交换机类型。transit 交换会在不同区域间路由流量。对于使用 OVN-Kubernetes 插件的集群，专用子网的 IP 地址都会被分配给附加到 transit 交换的任何逻辑端口。

注意

您可以将集群的加入、伪装和传输 CIDR 范围作为安装后任务更改。

OVN-Kubernetes 是 OpenShift Container Platform 4.14 及之后的版本中的默认网络供应商，内部使用以下 IP 地址子网范围：

- **V4JoinSubnet**: **100.64.0.0/16**
- **V6JoinSubnet**: **fd98::/64**
- **V4TransitSwitchSubnet**: **100.88.0.0/16**
- **V6TransitSwitchSubnet**: **fd97::/64**
- **defaultV4MasqueradeSubnet**:**169.254.0.0/17**
- **defaultV6MasqueradeSubnet**:**fd69::/112**

重要

前面的列表包括了 join、transfer 和 masquerade IPv4 和 IPv6 地址子网。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 地址子网范围。

其他资源

- 有关配置加入子网或传输子网的更多信息，请参阅配置 OVN-Kubernetes 内部 IP 地址子网。

4.1. 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。

> **注意**
>
> 您无法在创建集群后更改 Machine CIDR 范围。

默认值为 **10.0.0.0/16**。这个范围不得与任何连接的网络冲突。

其他资源

- Cluster Network Operator 配置

4.2. SERVICE CIDR

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

4.3. POD CIDR

在 pod CIDR 字段中，您必须为 pod 指定 IP 地址范围。

pod CIDR 与 **clusterNetwork** CIDR 和集群 CIDR 相同。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**。您可以在集群安装后扩展范围。

其他资源

- Cluster Network Operator 配置
- 配置集群网络范围

4.4. 主机前缀

在 **hostPrefix** 参数中，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 510 集群节点，以及每个节点的 510 个 pod IP 地址。

考虑将 **clusterNetwork.cidr** 参数设置为 **10.128.0.0/16** 的另一个示例，您要为集群定义完整的地址空间。这会为集群分配 65536 IP 地址池。然后，如果您将 **hostPrefix** 参数设置为 **/23**，则会为集群中的每个节点定义一个子网片段，其中 **/23** 是 **/16** 子网网络的一个子网。这会为每个节点分配 512 个 IP 地址，其中 2 个 IP 地址被保留用于网络和广播目的。以下示例计算使用这些 IP 地址图来确定您可以为集群创建的最大节点数：

```
65536 / 512 = 128
```

您可以使用 Red Hat OpenShift Network Calculator 计算集群的最大节点数。

4.5. 托管 CONTROL PLANE 的 CIDR 范围

要在 OpenShift Container Platform 上部署托管的 control plane，请使用以下所需的无类别域间路由 (CIDR)子网范围：

- **v4InternalSubnet**: 100.65.0.0/16 (OVN-Kubernetes)
- **clusterNetwork**: 10.132.0.0/14 (pod network)
- **serviceNetwork**: 172.31.0.0/16

如需有关 OpenShift Container Platform CIDR 范围定义的更多信息，请参阅"CIDR 范围定义"。