

NetScaler CPX 14.1

Contents

关于 NetScaler CPX	6
支持的功能	6
支持的平台	7
架构和流量流	9
NetScaler CPX 上的单一 IP 地址如何工作	9
源自 NetScaler CPX 实例的请求的流量流向	10
源自外部网络的请求的流量流向	11
NetScaler CPX 许可	13
NetScaler CPX 许可类型	13
NetScaler CPX 精简版	13
商用 NetScaler CPX	14
NetScaler CPX 实例支持的池化容量	14
NetScaler CPX 许可如何工作?	14
如何许可 NetScaler CPX?	15
开始使用基于 LAS 的 CPX 许可	16
排查 NetScaler CPX 许可问题	17
NetScaler CPX 对 Kubernetes 探测的支持	19
为 NetScaler CPX 配置 Kubernetes 探测	19
使用 CLI 配置探测	19
在 Docker 中部署 NetScaler CPX 实例	22
先决条件	22
从 Quay 下载 NetScaler CPX 映像	23
使用 docker run 命令部署 NetScaler CPX 实例	23
使用 docker run 命令部署更轻量级的 NetScaler CPX 版本	27
使用 Docker Compose 部署 NetScaler CPX 实例	28
将 NetScaler CPX 实例添加到 NetScaler Console	31

先决条件	31
使用环境变量将 NetScaler CPX 实例添加到 NetScaler Console	32
使用 Kubernetes ConfigMaps 将 NetScaler CPX 实例添加到 NetScaler Console	33
NetScaler CPX 许可证聚合器	37
NetScaler CPX 许可证聚合器的主要优势	37
部署拓扑	38
使用 Helm 图表 部署 NetScaler CPX 许可证聚合器	38
先决条件	38
使用 Helm chart 进行部署	39
安装 Helm chart	39
安装 NetScaler CPX License Aggregator 以管理带宽池许可证	39
安装 NetScaler CPX 许可证聚合器以管理 vCPU 许可证	40
安装 NetScaler CPX 许可证聚合器以管理多个许可证	41
配置 NetScaler CPX 以从 NetScaler CPX 许可证聚合器获取许可证	41
配置 NetScaler CPX	44
使用命令行界面配置 NetScaler CPX 实例	44
NetScaler CPX 中支持使用非默认密码	44
使用 NITRO API 配置 NetScaler CPX 实例	45
使用作业配置 NetScaler CPX 实例	45
在 NetScaler CPX 实例上配置 AppFlow	48
启用 AppFlow®	48
指定收集器	49
配置 AppFlow 操作	50
配置 AppFlow 策略	50
绑定 AppFlow 策略	50
使用配置文件配置 NetScaler CPX	52
静态配置文件的结构	52
NetScaler CPX 中的动态路由支持	55

使用 REST API 在 NetScaler CPX 上配置基于 BGP 的路由运行状况注入	55
配置 Docker 日志驱动程序	59
升级 NetScaler CPX 实例	60
先决条件	60
升级 NetScaler CPX 实例	61
在 NetScaler CPX 实例中使用通配符虚拟服务器	63
部署 NetScaler CPX 作为代理以启用东西向流量	65
在单个主机网络中部署 NetScaler CPX	70
在多主机网络中部署 NetScaler CPX	71
拓扑 1：NetScaler CPX 和后端服务器位于同一主机上；客户端位于不同网络上	71
拓扑 2：带物理服务器和客户端的 NetScaler CPX	74
拓扑 3：在不同主机上预置 NetScaler CPX 和服务器	76
部署 NetScaler CPX，直接访问网络	79
使用 ConfigMap 在 Kubernetes 中配置 NetScaler CPX	80
使用 ConfigMap 为 NetScaler CPX 配置自定义指标	85
配置自定义指标	85
使用 CLI 进行配置	85
使用 YAML 文件进行配置	87
部署 NetScaler CPX 作为 Kubernetes 节点的本地 DNS 缓存	89
在节点上部署 NetScaler CPX 作为 DNS 缓存	89
Kubernetes 主节点上的配置	90
Kubernetes 集群中工作节点上的配置	94
在 Google Compute Engine 上部署 NetScaler CPX 代理	96
先决条件	96
配置步骤	96
步骤 1：在 GCE VM 上安装 Docker	97
步骤 2：配置与 Docker 实例的远程 API 通信	108
步骤 3：安装 NetScaler CPX 映像	111

步骤 4：创建 NetScaler CPX 实例	112
步骤 5：通过 NetScaler Console 许可 NetScaler CPX	116
步骤 6: 在 NetScaler CPX 上配置负载均衡服务并验证配置	117
安装 NGINX Web 服务器	117
配置 NetScaler CPX 以进行负载均衡并验证负载到两个 Web 服务的分配	120
NetScaler CPX 故障排除	134
fix	143

关于 NetScaler CPX

NetScaler CPX 是一个基于容器的应用程序交付控制器，可以在 Docker 主机上进行预置。

NetScaler CPX 使客户能够利用 Docker 引擎功能，并为基于容器的应用程序使用 NetScaler 负载平衡和流量管理功能。您可以在 Docker 主机上将一个或多个 NetScaler CPX 实例部署为独立实例。

一个 NetScaler CPX 实例提供高达 1 Gbps 的吞吐量。

作为 NetScaler 的容器化形态，NetScaler CPX 很好地集成到 Kubernetes 环境中，并构成 NetScaler 云原生解决方案的组成部分。NetScaler 云原生解决方案可帮助您在 Kubernetes 环境中快速、灵活、高效地创建和交付软件应用程序。使用 NetScaler 云原生解决方案，您可以确保 Kubernetes 环境的企业级可靠性和安全性。有关更多信息，请参阅 [NetScaler 云原生解决方案](#)。

本文档假定您熟悉 Docker 及其工作原理。有关 Docker 的信息，请参阅 [<https://docs.docker.com>](https://docs.docker.com) 处的 Docker 文档。

支持的功能

NetScaler CPX 支持以下功能：

- 应用程序可用性
 - L4 负载平衡和 L7 内容切换
 - SSL 卸载
 - IPv6 协议转换
 - Microsoft SQL、MySQL 负载平衡
 - AppExpert 速率控制
 - 订阅者感知流量引导
 - 突发保护和优先级排队
 - 动态路由协议
- 应用程序加速
 - 客户端和服务端 TCP 优化
 - 缓存重定向
 - AppCompress

- AppCache
- 应用程序安全性
 - L7 重写和响应器
 - L4 DoS 防御
 - L7 DoS 防御
 - Web 应用程序防火墙 (WAF)。NetScaler CPX 支持其他 NetScaler 外形规格所支持的所有 WAF 功能。有关支持的 WAF 功能的信息，请参阅 [NetScaler Web 应用程序防火墙](#)。
- TCP 协议优化
 - 多路径 TCP
 - 二进制增加拥塞控制 (BIC) 和三次 TCP
- 简单可管理性
 - Web 日志记录
 - AppFlow®
 - NetScaler® 应用交付管理
 - 操作分析
- 应用程序优化
 - 集成缓存
- BGP 路由和路由健康注入 (RHI)

注意：

分配给 NetScaler CPX 设备接口（Linux 主机）的 Rx、Tx、GRO、GSO 和 LRO 等接口功能已禁用。即使在 NetScaler CPX 设备停止后，这些功能仍保持禁用状态。此外，此类接口的 MTU 已更改为 1500 字节。

支持的平台

NetScaler CPX 支持以下平台：

- 库伯内特斯
- 红帽 OpenShift

- 公有云
 - 亚马逊弹性 Kubernetes 服务 (EKS)
 - Azure 库伯内特斯 服务 (AKS)
 - Google 库伯内特斯 引擎 (GKE)
- 兰彻
- Pivotal 容器服务 (PKS)
- Docker 1.12 及更高版本

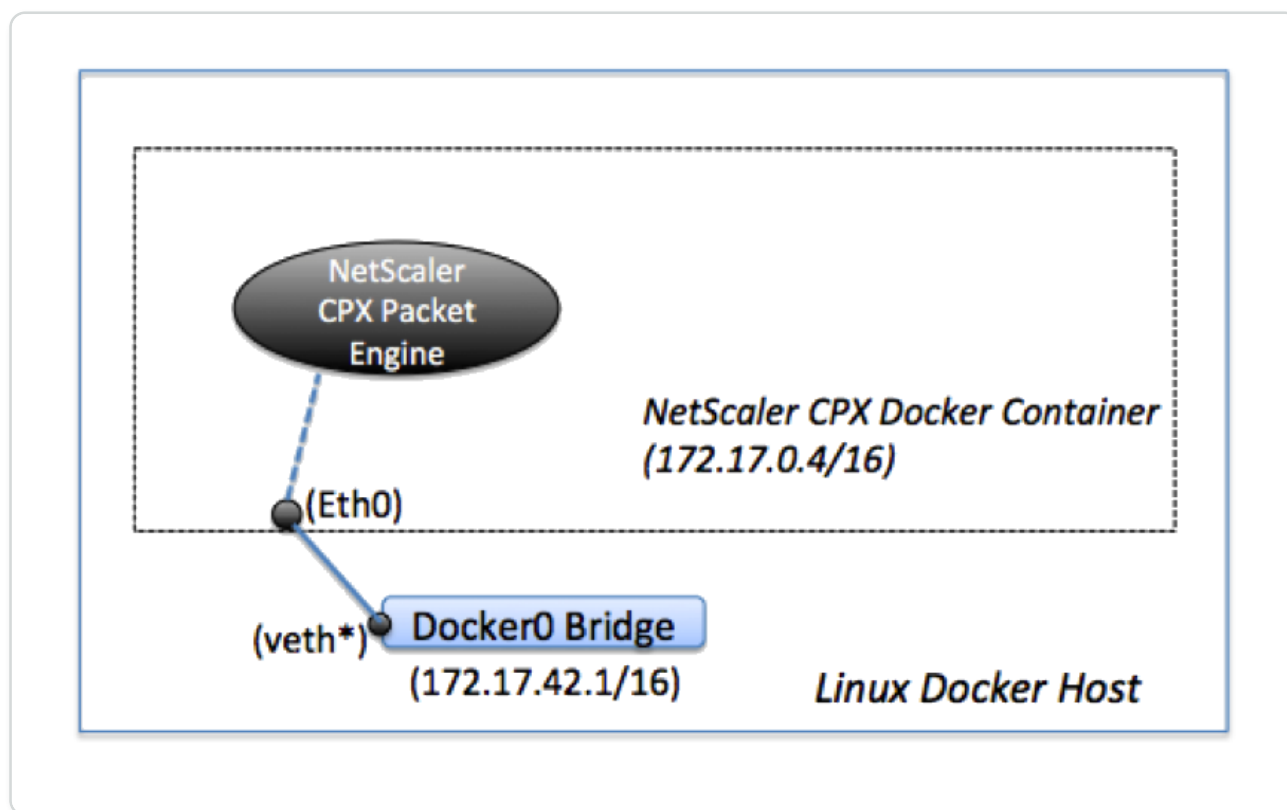
架构和流量流

本节介绍 NetScaler CPX 桥接模式架构和流量流。NetScaler CPX 也可以部署在主机模式下。

当您在 Docker 主机上预置 NetScaler CPX 实例时，Docker 引擎会在 CPX 实例上创建一个虚拟接口 eth0。此 eth0 接口直接连接到 docker0 网桥上的虚拟接口 (veth*)。Docker 引擎还会为 NetScaler CPX 实例分配一个 172.17.0.0/16 网络中的 IP 地址。

CPX 实例的默认网关是 docker0 网桥的 IP 地址，这意味着与 NetScaler CPX 实例的任何通信都是通过 Docker 网络完成的。所有从 docker0 网桥接收的入站流量都由 NetScaler CPX 实例上的 eth0 接口接收，并由 NetScaler CPX 数据包引擎处理。

下图说明了 Docker 主机上 NetScaler CPX 实例的架构。



NetScaler CPX 上的单一 IP 地址如何工作

常规的 NetScaler MPX 或 VPX 设备需要至少三个 IP 地址才能运行：

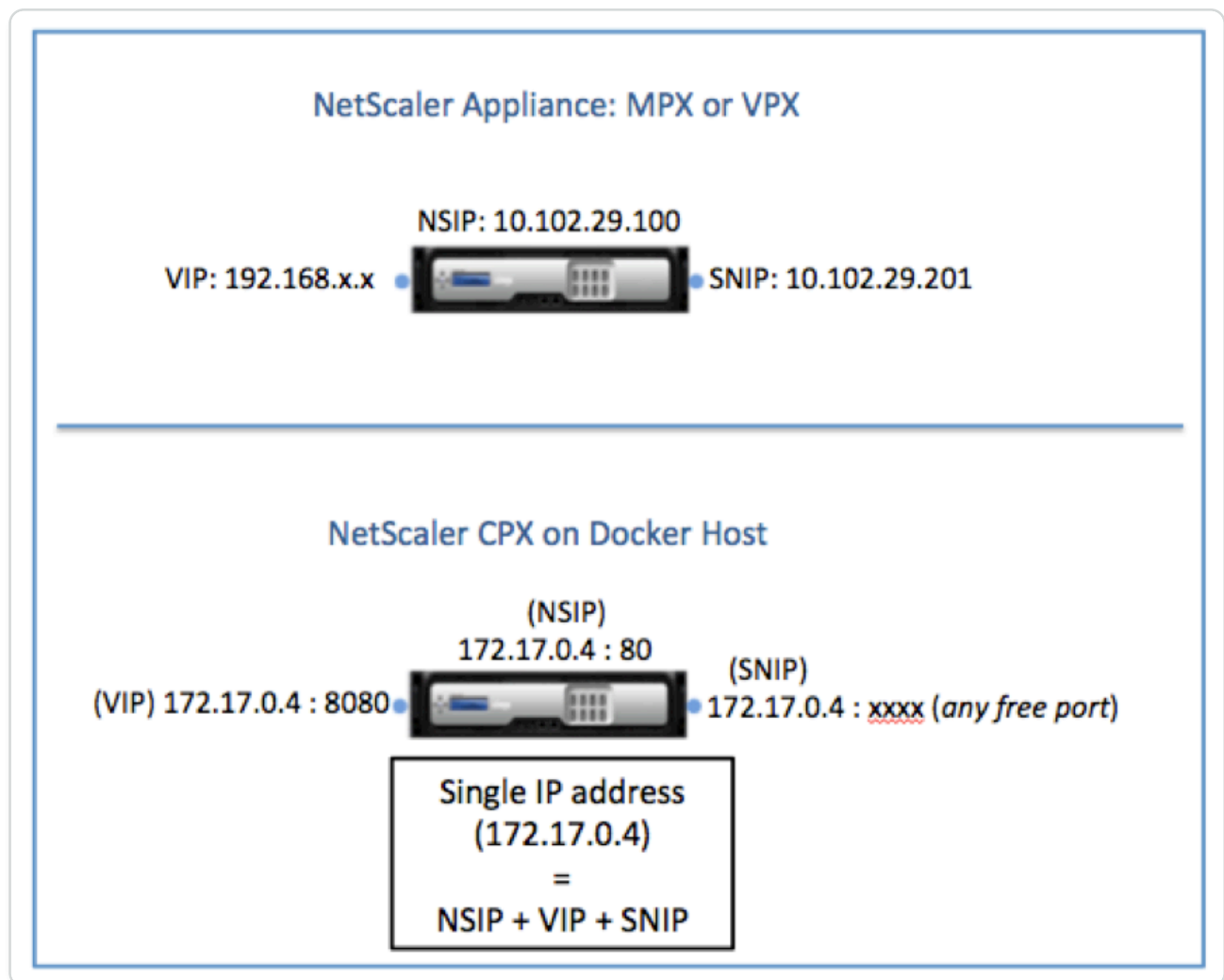
- 称为 NetScaler IP (NSIP) 地址的管理 IP 地址
- 用于与服务器场通信的子网 IP (SNIP) 地址

- 用于接受客户端请求的虚拟服务器 IP (VIP) 地址

NetScaler CPX 实例使用一个单一 IP 地址运行，该地址用于管理和数据流量。

在预置期间，Docker 引擎仅为 NetScaler CPX 实例分配一个私有 IP 地址（单一 IP 地址）。NetScaler 实例的三个 IP 功能复用到一个 IP 地址上。此单一 IP 地址使用不同的端口号来充当 NSIP、SNIP 和 VIP。

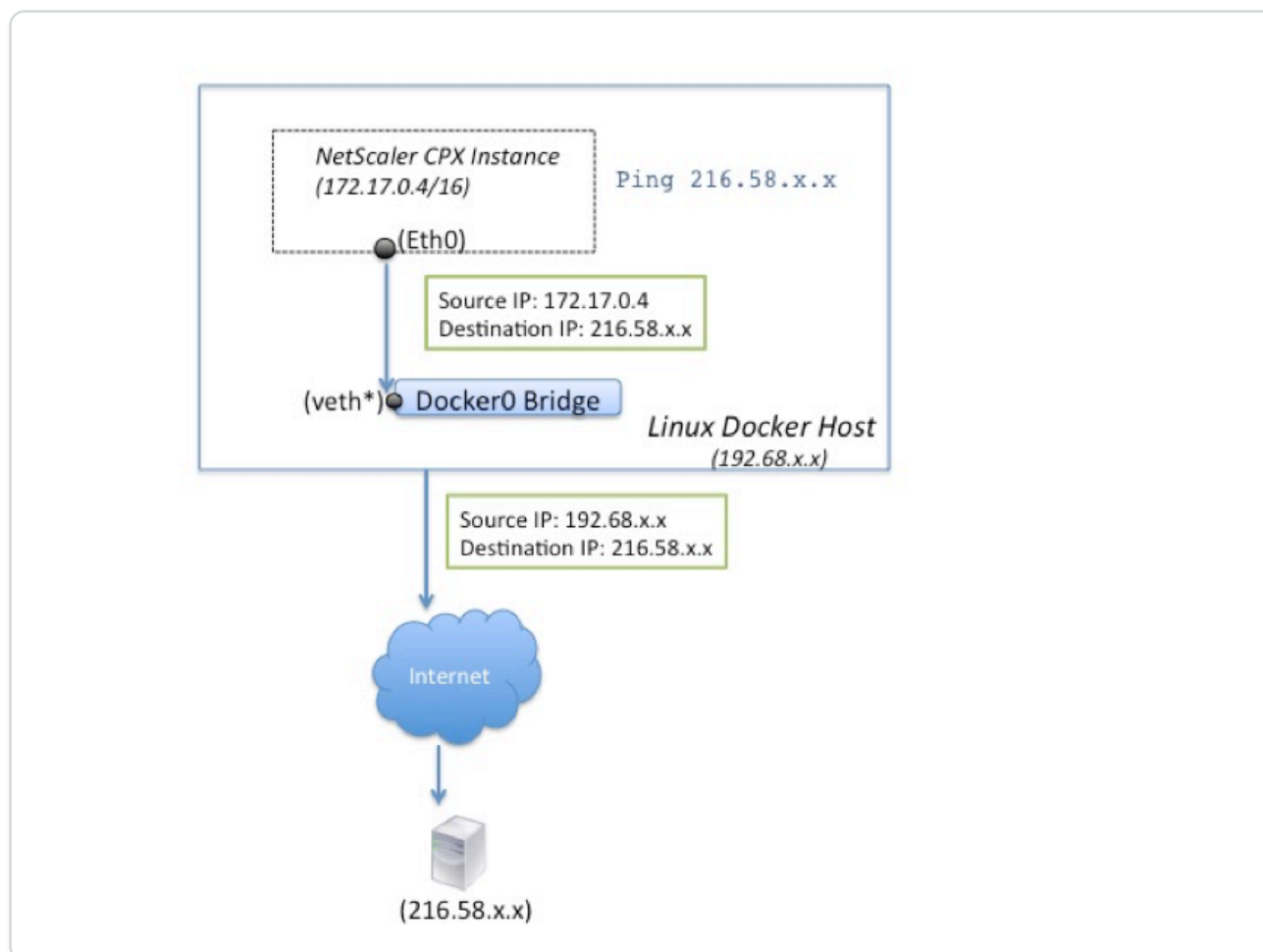
下图说明了如何使用单一 IP 地址来执行 NSIP、SNIP 和 VIP 的功能。



源自 NetScaler CPX 实例的请求的流量流向

Docker 隐式配置 IP 表和 NAT 规则，以将源自 NetScaler CPX 实例的流量导向 docker0 IP 地址。

下图说明了源自 NetScaler CPX 实例的 ping 请求如何到达目标。



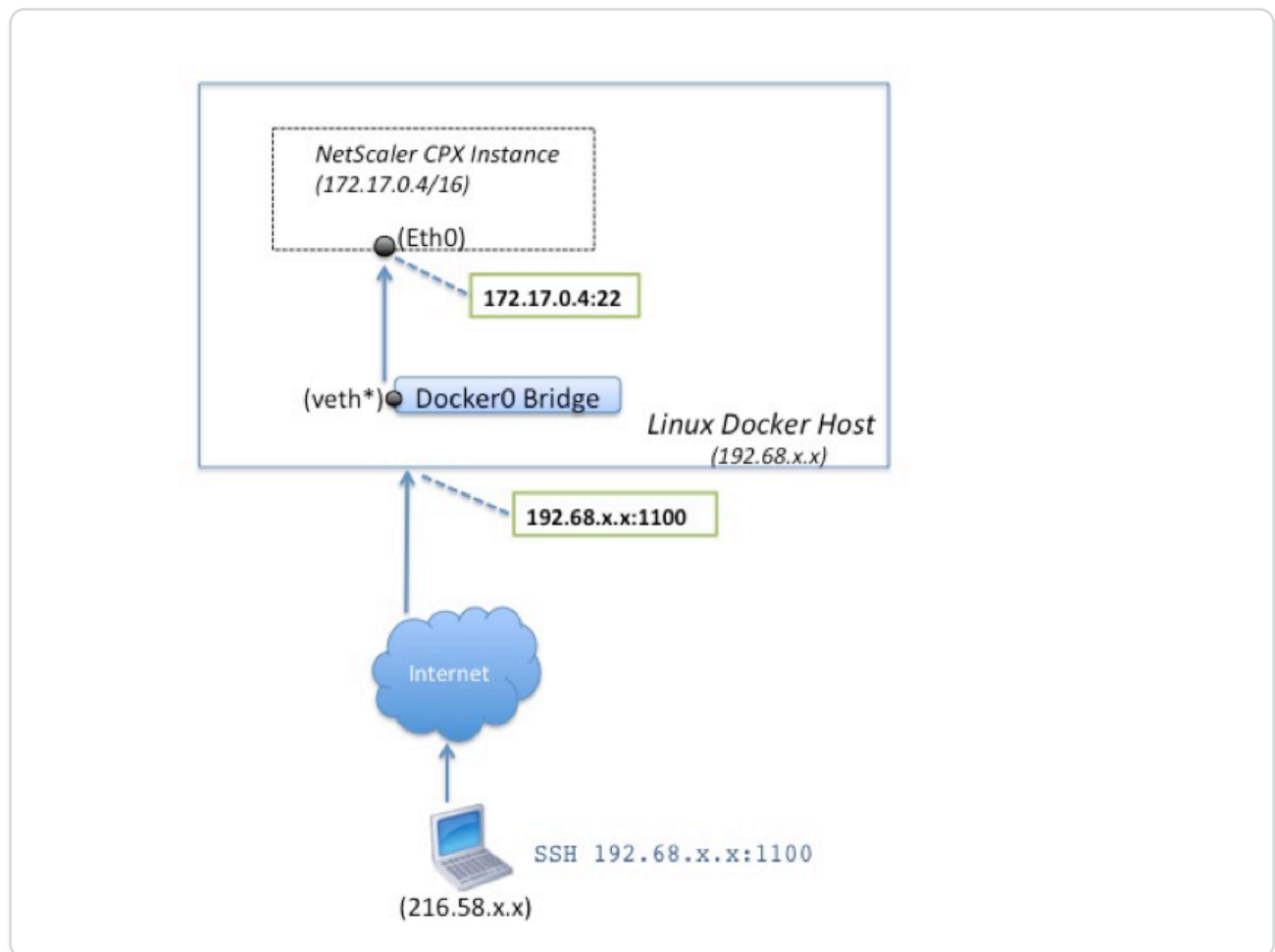
在此示例中，ping 请求由 eth0 接口上的数据包引擎发送，源 IP 地址为 NetScaler CPX IP 地址 (172.17.0.4)。然后，Docker 主机执行网络地址转换 (NAT)，以将主机 IP 地址 (192.68.x.x) 添加为源 IP 地址，并将请求发送到目标 (216.58.x.x)。来自目标 IP 地址的响应沿相同路径反向返回。Docker 主机对响应执行 NAT，并将响应转发到 eth0 接口上的 NetScaler CPX 实例。

源自外部网络的请求的流量流向

为了启用外部通信，在预置 NetScaler CPX 时，您必须设置参数，以便 Docker 暴露某些端口，例如 80、22 和您想要的任何其他端口。如果在预置期间未设置任何要暴露的端口，则必须在 Docker 主机上配置 NAT 规则以使这些端口可用。

源自 Internet 的客户端请求由 Docker 主机接收，然后 Docker 主机执行端口地址转换 (PAT)，以将公共 IP 地址和端口映射到 NetScaler CPX 实例的单个 IP 地址和端口，并将流量转发到该实例。

下图显示了 Docker 主机如何执行端口地址转换，以将流量导向 NetScaler CPX 的单个 IP 地址和端口。



在此示例中，Docker 主机 IP 地址为 192.68.x.x，NetScaler CPX 实例的单个 IP 地址为 172.17.0.4。NetScaler CPX 实例的 SSH 端口 22 映射到 Docker 主机上的端口 1100。来自客户端的 SSH 请求在 IP 地址 192.68.x.x 的端口 1100 上接收。Docker 主机执行端口地址转换，将此地址和端口映射到端口 22 上的单个 IP 地址 172.17.0.4，并转发客户端请求。

NetScaler CPX 许可

[NetScaler CPX](#) 是一种基于容器的应用程序交付控制器，可在 Docker 主机上预置，用于对基于微服务的应用程序进行负载平衡。您需要获得许可的 NetScaler CPX 才能获得更好的应用程序交付性能。NetScaler CPX 支持传统池许可和新的 [Citrix Platform](#) 或 [Universal Hybrid Multi-Cloud](#) 许可。所有 NetScaler CPX 实例仅通过 NetScaler Console（以前称为 NetScaler ADM）获得许可。

NetScaler Console 提供本地部署和云服务两种形式。您可以使用 NetScaler Console 管理所有 NetScaler 外形规格的池容量或通用混合多云 (UHMC) 或 Citrix 平台许可证 (CPL) 许可。

有关本地部署 NetScaler Console 的信息，请参阅 [本地部署 NetScaler Console](#)。有关 NetScaler Console 服务的信息，请参阅 [NetScaler Console 服务](#)。

NetScaler CPX 许可类型

支持以下类型的 NetScaler CPX 许可：

- NetScaler CPX CPX Express - 免费、以开发人员为中心的 Kubernetes NetScaler
- 商用 NetScaler CPX – 生产就绪、企业支持代理

NetScaler CPX 精简版

NetScaler CPX Express 是一种适用于本地部署和云部署的免费软件版本。当您从 [Quay](#) 存储库下载 NetScaler CPX 实例时，NetScaler CPX Express 许可默认应用于 NetScaler CPX 实例。NetScaler CPX Express 许可提供所有 NetScaler CPX 功能。有关 CPX 支持的功能的信息，请参阅 [About NetScaler CPX](#)。NetScaler CPX Express 提供 NetScaler 社区支持，但不提供企业支持。

NetScaler CPX Express 许可提供 100 Mbps 带宽容量。您必须为 NetScaler CPX 实例购买许可，以获得更高的容量、用于生产部署并获得企业支持。

注意：

从 NetScaler CPX 14.1 版本 17.x 开始，NetScaler CPX Express 许可的带宽从 20 Mbps 增加到 100 Mbps。增加的带宽使 Kubernetes 开发人员能够试用 NetScaler 代理。

商用 NetScaler CPX

在 April 15, 2026 之后，LAS 将是激活和许可 NetScaler 实例的唯一方式，支持 NetScaler Flexed 许可 (CPL/UHMC)、传统 NetScaler 带宽池许可。在 LAS 转换后，NetScaler CPX 将仅支持 NetScaler Flexed 许可 (CPL/UMC) 或传统 NetScaler 池带宽许可。有关更多信息，请参阅 [过渡到弹性许可](#) 以了解更多详细信息。

NetScaler CPX 实例支持的池化容量

产品	最大带宽	最小带宽	最小实例数	最大实例数	最小带宽单位
NetScaler CPX	40000 注意：这取决于 CPU 频率、代次等。	20 兆比特/秒	1	16	10 兆比特/秒

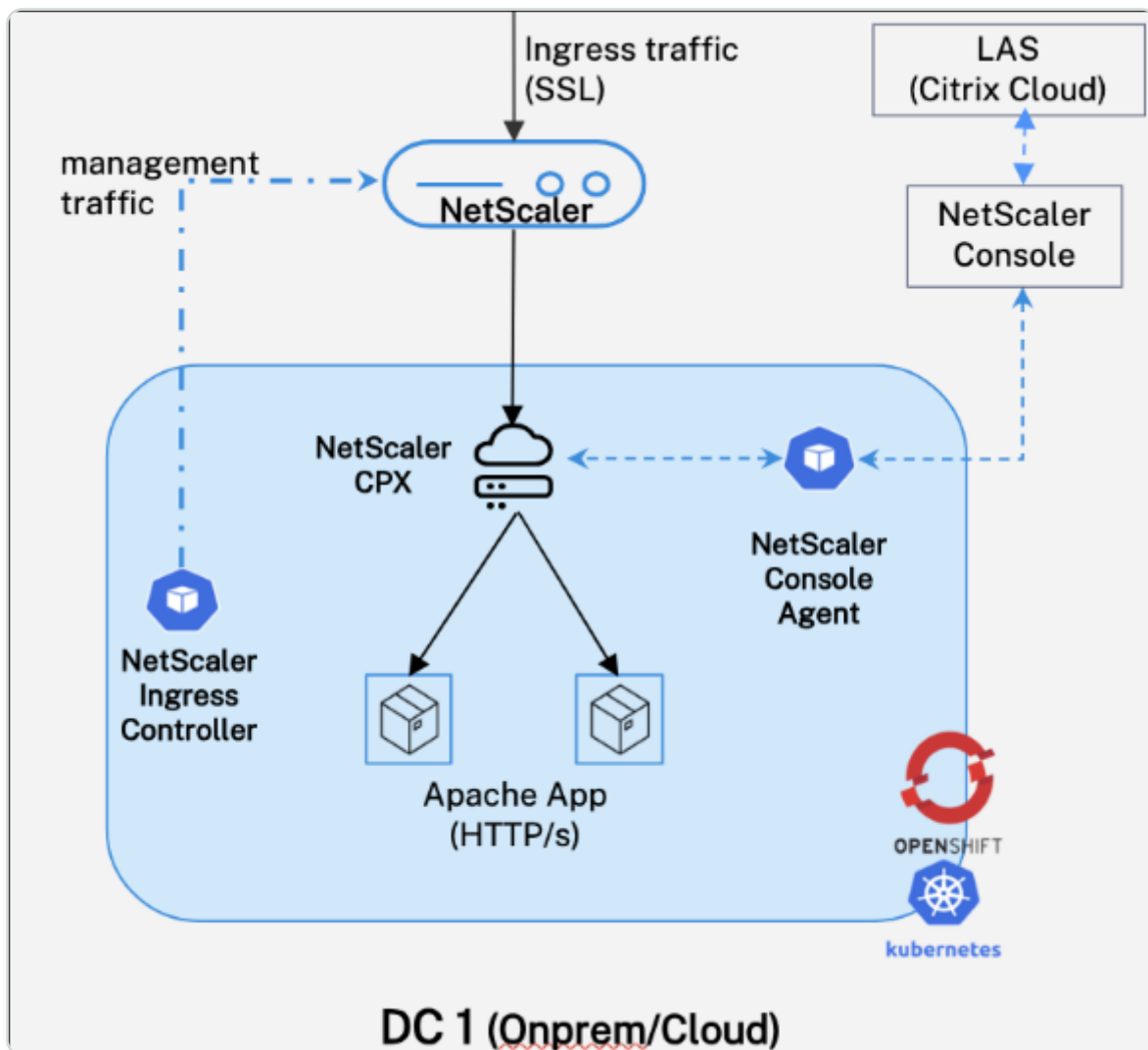
注意：

Citrix 目前正在为基于公共云的产品开发基于 NetScaler CPX 消耗或按需增长的许可模型。一旦准备就绪，它将在公共云市场中提供使用。

NetScaler CPX 许可如何工作？

NetScaler CPX 池化/CPL/UHMC 许可： 一个公共许可证池，您的 NetScaler CPX 实例可以从中签出一个实例许可证，并仅获取其所需的带宽。当实例不再需要这些资源时，它会将它们退回公共池，使这些资源可供其他需要的实例使用。

NetScaler CPX 行为： 单个 NetScaler CPX 实例签出高达 1 Gbps 的吞吐量时，仅从实例池签出，而不从带宽许可证池签出。NetScaler CPX 以这种方式运行，直至带宽利用率达到 1 Gbps。例如，如果 NetScaler CPX 实例消耗 200 Mbps 带宽，它将使用实例许可证池，而不是带宽池。但是，如果 NetScaler CPX 实例消耗 1200 Mbps 的吞吐量，则前 1000 Mbps 从实例池中使用，剩余的 200 Mbps 从带宽池中消耗。



NetScaler CPX 许可工作流程：

1. 设置 NetScaler 代理服务或本地 NetScaler 代理与 Citrix Cloud 配合进行 LAS 许可。
2. 部署 Console Container 代理并在 NetScaler 代理服务或本地 NetScaler 代理上检查其状态。
3. 配置 NetScaler CPX 许可并部署 NetScaler CPX。
 - a. 启动时，NetScaler CPX 会向 Console Container 代理发送许可证注册请求，后者将其转发给 NetScaler 代理服务或本地 NetScaler 代理。
 - b. NetScaler 代理服务或本地 NetScaler 代理注册 NetScaler CPX 并分配带宽。
 - c. 如果池化/UHMC/CPL 许可证 blob 可用，则授予许可证。

如何许可 NetScaler CPX?

NetScaler CPX 通过 NetScaler 代理服务或本地部署获得许可。要许可 Kubernetes 集群中的 NetScaler CPX 实例，请部署单个 Console 容器代理。每个集群一个代理许可该集群中的所有 NetScaler CPX 实例。

开始使用基于 LAS 的 CPX 许可

1. 先决条件：必须启用 [NetScaler Console Service](#) 或 [NetScaler Console](#) 本地部署的基于 LAS 的许可。
2. 在部署了 NetScaler CPX 实例的 Kubernetes 集群中部署基于容器的 Console 代理。
 - a. 以下是为 NetScaler Console 服务客户部署基于容器的 Console 代理的步骤：
 - i. 使用 [HELM chart](#)
 - ii. 使用 [OpenShift Operator](#)
 - b. 以下是为 NetScaler Console 本地部署客户部署基于容器的 Console 代理的步骤。
 - i. 使用 [YAMLs](#)。
 - ii. 使用 [OpenShift Operator](#)。
3. 升级到 NetScaler CPX 14.1–51.x 或更高版本以支持基于 LAS 的许可。有关最新的 NetScaler CPX 部署，请参阅 CPX HELM chart。仅从 CPX HELM chart 的 `values.yaml` 更新以下强制变量以支持基于 LAS 的许可。
 - a. `licensingMode` : `las`
 - b. `ADMIP`: `<Console Container agent Service IP>` - 从 NetScaler Console 本地部署或服务的 **Infrastructure > Instances Dashboard > Agents** 部分复制容器代理 IP 地址。

或

`ADMIP`: `<Console Container agent ServiceName.Namespace>` - 从 NetScaler Console 本地部署或服务的 **Infrastructure > Instances Dashboard > Agents** 部分复制容器代理 `HOSTNAME`，并将其与部署容器代理的命名空间一起附加。
 - c. `loginSecret` : NetScaler Console 容器代理登录凭据详细信息。
 - i. 如果是基于 YAML/HELM 的部署，您必须根据从 NetScaler Console 本地部署或服务生成控制台代理 YAML 时提供的应用程序 ID 和密码详细信息创建 Kubernetes secret。
 - ii. 在基于 OpenShift Operator 的部署中，您必须已经创建了 NetScaler Console 登录 secret（请参阅 `loginSecret` 字段）。您可以在此处使用相同的 secret 名称。
 - iii. `bandWidth`: 以 Mbps 为单位的值。例如，500 Mbps。

```

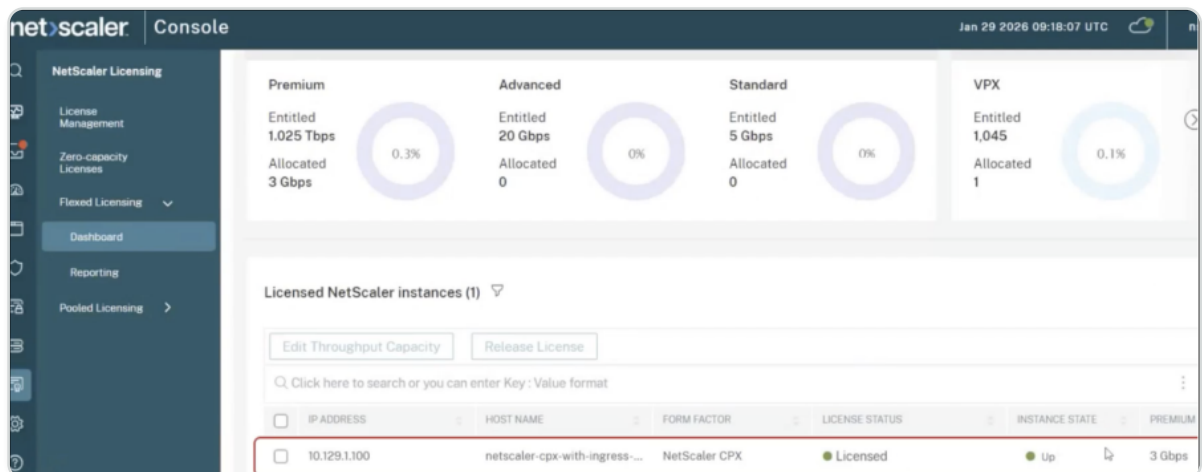
master netscaler-helm-charts / netscaler-cpx-with-ingress-controller / values.yaml

Blame 312 lines (277 loc) · 8.4 KB

# NetScaler ADM/License Server config details
ADMSettings:
  licenseServerIP:
  licenseServerPort: 27000
  ADMIP: ""
  loginSecret: ""
  bandwidthLicense: false
  bandwidth: 1000 #bandwidth value should be in Mbps
  vCPULicense: false
  cpxCores: 1
  platform: false
  licenseEdition: PLATINUM
  licensingMode: "" # "pool" or "las"
# For licensingmode "las", set the ADMIP, loginSecret, licenseEdition and bandwidth

```

- d. NetScaler CPX 许可成功后，NetScaler CPX 实例开始显示在 NetScaler Console 本地部署或服务仪表板中。



排查 NetScaler CPX 许可问题

- 验证 NetScaler CPX 许可。
 - 使用 `oc exec -it <CPX POD name> --bash` 登录到 NetScaler CPX，然后运行 `cli_script.sh "sh capacity"` 以查看实际带宽值。
 - 运行 `cli_script.sh "sh license"` 以检查 `Licensing mode: LAS` 值。
- 查找 NetScaler Console 服务或本地部署中的已许可 NetScaler CPX 实例。

- 在 NetScaler Console 服务或本地部署 GUI 上，导航到 **NetScaler Licensing > Flexed licensing > Dashboard**，并搜索 NetScaler Console 外形规格。

NetScaler CPX 对 Kubernetes 探测的支持

Kubernetes 通过启动探测、存活探测和就绪探测来管理容器生命周期，以确保应用程序正确初始化并保持健康。NetScaler CPX 现在除了支持现有的启动探测和存活探测外，还支持就绪探测。此支持确保 Kubernetes 仅在 NetScaler CPX 完全运行且（如果已配置）已成功完成许可证签出时才将流量路由到它。

通过使用这些探测，您可以提高应用程序可用性，并确保及时重新创建失败或无响应的实例。

其中一些好处包括：

- 增强的流量管理：确保 NetScaler CPX 仅在准备好处理请求时才接收流量。
- 运行可靠性：自动重启不健康的容器以从故障中恢复。
- 许可证验证：当使用 `LAS_CPX_BANDWIDTH` 参数时，延迟就绪状态直到许可证签出成功。
- 精确初始化：协调从容器启动到活动健康监控的过渡。

注意：

使用就绪探测需要 NetScaler CPX `14.1-72.x` 或更高版本。有关探测机制的详细信息，请参阅 Kubernetes 官方文档。

为 NetScaler CPX 配置 Kubernetes 探测

要为 NetScaler CPX 配置探测，请在 Kubernetes 部署 YAML 文件中定义探测参数。

使用 CLI 配置探测

编辑您的部署 YAML 文件，以包含 Kubernetes 用于验证 NetScaler CPX 容器状态的特定 `exec` 命令。

配置启动探测：添加以下代码片段，以确保 Kubernetes 等待初始化过程完成。NetScaler CPX 通过创建 `{{tmp/cpx_started}}` 文件来表示成功启动。配置启动探测以监视此路径，以确保在存活检查开始之前完成初始化。

```
startupProbe:

  exec:

    command:

      - ls

      - /tmp/cpx_started

  initialDelaySeconds: 30

  periodSeconds: 5

  failureThreshold: 20

  successThreshold: 1

  timeoutSeconds: 1
```

配置就绪探针：添加以下代码片段，以定义 Pod 何时准备好接受流量。CPX 在准备好处理流量时创建 `/tmp/cpx_ready` 文件。如果您配置了 `LAS_CPX_BANDWIDTH` 参数，CPX 会在成功进行许可证检查后创建此文件。否则，它会在启动过程完成后出现。Kubernetes 使用此文件进行就绪探针，以确定何时将流量路由到 Pod。

```
readinessProbe:

  exec:

    command:

      - /bin/ls

      - /tmp/cpx_ready

  failureThreshold: 15

  initialDelaySeconds: 60

  periodSeconds: 30
```

```
successThreshold: 1
```

```
timeoutSeconds: 1
```

配置活跃度探针：添加以下代码片段，以在运行时持续监控容器的运行状况。在部署 YAML 中定义活跃度探针，以在运行时监控 CPX 运行状况。如果探针失败，Kubernetes 会重新启动容器以从不健康状态中恢复。此机制通过自动重新创建无响应的 CPX 实例来确保应用程序可用性。

```
livenessProbe:
```

```
  exec:
```

```
    command:
```

```
      - /bin/ping
```

```
      - -c 1
```

```
      - 192.0.0.1
```

```
  initialDelaySeconds: 10
```

```
  periodSeconds: 5
```

```
  failureThreshold: 3
```

```
  successThreshold: 1
```

```
  timeoutSeconds: 1
```

在 Docker 中部署 NetScaler CPX 实例

NetScaler CPX 实例作为 Docker 镜像文件在 Quay 容器注册表中提供。要部署实例，请从 Quay 容器注册表下载 NetScaler CPX 镜像，然后使用 `docker run` 命令或 Docker compose 工具部署实例。

先决条件

确保：

- Docker 主机系统至少具有：
 - 1 个 CPU
 - 2 GB 内存

注意

为了获得更好的 NetScaler CPX 性能，您可以定义希望 NetScaler CPX 实例启动的处理引擎数量。对于您添加的每个额外的处理引擎，请确保 Docker 主机包含等量的 vCPU 数量和 GB 内存。例如，如果您想添加 4 个处理引擎，Docker 主机必须包含 4 个 vCPU 和 4 GB 内存。

- Docker 主机系统运行 Linux Ubuntu 14.04 或更高版本。
- 主机系统上安装了 Docker 1.12 版。有关在 Linux 上安装 Docker 的信息，请参阅 [Docker 文档](#)。
- Docker 主机具有 Internet 连接。

注意：NetScaler CPX 在 ubuntu 16.04.5 版本、内核版本 4.4.0-131-generic 上运行时存在问题。因此，不建议在 ubuntu 16.04.5 版本、内核版本 4.4.0-131-generic 上运行 NetScaler CPX。

注意：以下 kubelet 和 kube-proxy 版本存在一些安全漏洞，不建议将 NetScaler CPX 与这些版本一起使用：

- kubelet/kube-proxy v1.18.0-1.18.3

- kubelet/kube-proxy v1.17.0-1.17.6
- kubelet/kube-proxy <=1.16.10

有关如何缓解此漏洞的信息，请参阅[缓解此漏洞](#)。

从 Quay 下载 NetScaler CPX 映像

您可以使用 `docker pull` 命令从 Quay 容器注册表下载 NetScaler CPX 映像并将其部署到您的环境中。使用以下命令从 Quay 容器注册表下载 NetScaler CPX 映像：

```
docker pull quay.io/citrix/citrix-k8s-cpx-ingress:13.0-xx.xx
```

例如，如果您要下载版本 13.0-64.35，请使用以下命令：

```
docker pull quay.io/citrix/citrix-k8s-cpx-ingress:13.0-64.35
```

使用以下命令验证 NetScaler CPX 映像是否已安装在 Docker 映像中：

```
root@ubuntu:~# docker images | grep 'citrix-k8s-cpx-ingress'
```

quay.io/citrix/citrix-k8s-cpx-ingress	13.0-64.35
952a04e73101	2 months ago 469 MB

您可以从 [Quay 容器注册表](#) 部署最新的 NetScaler CPX 映像。

使用 docker run 命令部署 NetScaler CPX 实例

在主机上，您可以使用已加载到主机的 NetScaler CPX Docker 映像，在 Docker 容器中安装 NetScaler CPX 实例。使用 `docker run` 命令，安装具有默认 NetScaler CPX 配置的 NetScaler CPX 实例。

使用以下 **docker run** 命令在 Docker 容器上安装 NetScaler CPX 实例：

```
docker run -dt -P --privileged=true --net=host -e NS_NETMODE="HOST" -e  
CPX_CORES=<number of cores> --name <container_name> --ulimit core=-1 -e  
CPX_NW_DEV='<INTERFACES>' -e CPX_CONFIG='{ "YIELD": "NO" }' -e  
LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT> -e PLATFORM=CP1000 -v
```

```
<host_dir>:/cpx --name <cpx_container_name> <REPOSITORY>:<CPX-  
IMAGE_TAG_IN_QUAY>
```

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e  
CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"No"}' -e  
LS_IP=10.102.38.134 -e PLATFORM=CP1000 -v /var/cpx:/cpx --name cpx_host  
quay.io/citrix/citrix-k8s-cpx-ingress:14.1-x.x
```

此示例基于 NetScaler CPX Docker 映像创建了一个名为 `mycpx` 的容器。

`-P` 参数是强制性的。它告诉 Docker 映射 NetScaler CPX Docker 映像中公开的端口。这意味着将端口 9080、22、9443 和 161/UDP 映射到从用户定义范围中随机选择的 Docker 主机上的端口。执行此映射是为了避免冲突。如果您以后在同一 Docker 主机上创建多个 NetScaler CPX 容器。端口映射是动态的，并在每次容器启动或重新启动时设置。端口的使用方式如下：

- 9080 用于 HTTP
- 9443 用于 HTTPS
- 22 用于 SSH
- 161/UDP 用于 SNMP。

如果要进行静态端口映射，请使用 `-p` 参数手动设置。

`--privileged=true` 选项用于在特权模式下运行容器。如果您在主机部署模式下运行 NetScaler CPX，则需要向 NetScaler CPX 提供所有系统特权。

要在桥接模式下运行具有单个或多个核心的 NetScaler CPX，您可以使用 `--cap-add=NET_ADMIN` 选项而不是特权模式。`--cap-add=NET_ADMIN` 选项使您能够以完整的网络特权运行 NetScaler CPX 容器。使用 `--cap-add=NET_ADMIN` 选项时，请在 `docker run` 命令中使用 `--sysctl kernel.shmmax=1073741824 --sysctl net.ipv6.conf.default.accept_dad=0 --sysctl kernel.core_pattern=/var/crash/core.%e.%p.%s` 选项手动执行其他系统控制设置。这些额外的系统控制设置在任何非特权模式下都不会自动完成。

`**--net=host` 是一个标准的 `docker run` 命令选项，它指定容器在主机网络堆栈中运行并可以访问所有网络设备。

注意

如果您在桥接或无网络模式下运行 NetScaler CPX，请忽略此选项。

`-e NS_NETMODE="HOST"` 是一个 NetScaler CPX 特定的环境变量，允许您指定 NetScaler CPX 以主机模式启动。一旦 NetScaler CPX 以主机模式启动，它会在主机上配置 4 条默认的 iptables 规则，用于管理访问 NetScaler CPX。它使用以下端口：

- 9995 用于 HTTP
- 9996 用于 HTTPS
- 9997 用于 SSH
- 9998 用于 SNMP

如果要指定不同的端口，可以使用以下环境变量：

- `-e NS_HTTP_PORT=`
- `-e NS_HTTPS_PORT=`
- `-e NS_SSH_PORT=`
- `-e NS_SNMP_PORT=`

注意

如果 NetScaler CPX 在桥接或无网络模式下运行，请忽略此环境变量。

`-e CPX_CORES` 是一个可选的 NetScaler CPX 特定环境变量。您可以通过定义希望 NetScaler CPX 容器启动的处理引擎数量来提高 NetScaler CPX 实例的性能。

注意

- 对于您添加的每个额外的处理引擎，请确保 Docker 主机包含等量的 vCPU 和以 GB 为单位的内存。例如，如果您想添加 4 个处理引擎，则 Docker 主机必须包含 4 个 vCPU 和 4 GB 内存。
- NetScaler CPX 支持 1 到 16 个核心。

`-e EULA = yes` 是一个强制性的 NetScaler CPX 特定环境变量，需要它来验证您已阅读并理解以下位置提供的最终用户许可协议 (EULA)：[Citrix 许可系统组件](#)。

`-e PLATFORM=CP1000` 参数指定 NetScaler CPX 许可证类型。

如果您在主机网络中运行 Docker，可以使用 `-e CPX_NW_DEV` 环境变量为 NetScaler CPX 容器分配专用网络接口。您需要定义以空格分隔的网络接口。您定义的网络接口将由 NetScaler CPX 容器持有，直到您卸载 NetScaler CPX 容器。当 NetScaler CPX 容器被配置时，所有分配的网络接口都将添加到 NetScaler 网络命名空间中。

注意

如果您在桥接网络中运行 NetScaler CPX，您可以更改容器网络，例如，为容器配置另一个网络连接或删除现有网络。然后，请确保重新启动 NetScaler CPX 容器以使用更新后的网络。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e PLATFORM=CP1000 --
name cpx_host cpx:13.0-x.x
```

`-e CPX_CONFIG` 是一个 NetScaler CPX 特定环境变量，可让您控制 NetScaler CPX 容器的吞吐量性能。当 NetScaler CPX 未收到任何传入流量进行处理时，它会在空闲期间让出 CPU，从而导致吞吐量性能低下。在这种情况下，您可以使用 `CPX_CONFIG` 环境变量来控制 NetScaler CPX 容器的吞吐量性能。您需要以 JSON 格式向 `CPX_CONFIG` 环境变量提供以下值：

- 如果您希望 NetScaler CPX 容器在空闲场景中让出 CPU，请定义 `{"YIELD" : "Yes"}`
- 如果您希望 NetScaler CPX 容器在空闲场景中避免让出 CPU，以便获得高吞吐量性能，请定义 `{"YIELD" : "No"}`

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD": "No"}' -e PLATFORM=CP1000
--name cpx_host cpx:13.0-x.x
```

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD": "Yes"}' -e PLATFORM=CP1000
--name cpx_host cpx:13.0-x.x
```

`-v` 参数是一个可选参数，用于指定 NetScaler CPX 挂载目录 `/cpx` 的挂载点。挂载点是主机上的一个目录，您可以在其中挂载 `/cpx` 目录。`/cpx` 目录存储日志、配置文件、SSL 证书和核心转储文件。在此示例中，挂载点是 `/var/cpx`，NetScaler CPX 挂载目录是 `/cpx`。

如果您购买了许可证或拥有评估许可证，您可以使用 `-e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT>` 参数，通过 `docker run` 命令将许可证上传到许可证服务器并指定许可证服务器位置。在这种情况下，您无需接受 EULA。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e  
CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"No"}' -e LS_IP=10.102.38.134 -e  
PLATFORM=CP1000 --name cpx_host cpx:13.0-x.x
```

其中：

- `LS_IP_ADDRESS` 是许可证服务器的 IP 地址。
- `LS_PORT` 是许可证服务器的端口。

您可以使用以下命令查看系统上运行的映像以及映射到标准端口的端口：`docker ps`

使用 `docker run` 命令部署更轻量级的 NetScaler CPX 版本

NetScaler 提供了一个更轻量级的 NetScaler CPX 版本，它消耗更少的运行时内存。更轻量级的 NetScaler CPX 版本可以作为 sidecar 部署在服务网格部署中。

更轻量级的 NetScaler CPX 版本支持以下功能：

- 应用程序可用性
 - L4 负载均衡和 L7 内容切换
 - SSL 卸载
 - IPv6 协议转换
- 应用程序安全
 - L7 重写和响应器
- 简单可管理性
 - Web 日志记录
 - AppFlow®

要实例化 NetScaler CPX 的精简版，请在执行 `Docker run` 命令时设置 `NS_CPX_LITE` 环境变量。

```
docker run -dt -P --privileged=true -e NS_CPX_LITE=1 -e EULA=yes --name  
<container_name> --ulimit core=-1 <REPOSITORY>:<TAG>
```

以下示例基于 NetScaler CPX 映像创建一个轻量级容器。

```
docker run -dt -P --privileged=true -e NS_CPX_LITE=1 -e EULA=yes --name
lightweight --ulimit core=-1 cpx:latest
```

默认情况下，NetScaler CPX 精简版上使用 `newnslog` 的日志记录处于禁用状态。要启用它，您必须在启动 NetScaler CPX 精简版时将 `NS_ENABLE_NEWNSLOG` 环境变量设置为 1。

以下示例展示了如何在部署 NetScaler CPX 精简版时使用 `newnslog` 启用日志记录。

```
docker run -dt --privileged=true --ulimit core=-1 -e EULA=yes -e
NS_CPX_LITE=1 -e NS_ENABLE_NEWNSLOG=1 cpx:<tag>
```

注意：

CPX 精简版仅支持单核 (CPX_CORES=1)。

使用 Docker Compose 部署 NetScaler CPX 实例

您可以使用 Docker 的 Compose 工具来预置单个 NetScaler CPX 实例或多个 NetScaler CPX 实例。要使用 Docker Compose 预置 NetScaler CPX 实例，您必须首先编写一个 Compose 文件。此文件指定 NetScaler CPX 映像、要为 NetScaler CPX 实例打开的端口以及 NetScaler CPX 实例的权限。

重要

确保您已在主机上安装 Docker Compose 工具。

要预置多个 NetScaler CPX 实例：

1. 编写一个 Compose 文件，其中：

- `<service-name>` 是您要预置的服务名称。
- `image:<repository>:<tag>` 表示 NetScaler CPX 映像的存储库和版本。
- `privileged: true` 为 NetScaler CPX 实例提供所有根权限。
- `cap_add` 为 NetScaler CPX 实例提供网络权限。
- `<host_directory_path>` 表示您要为 NetScaler CPX 实例挂载的 Docker 主机上的目录。
- `<number_processing_engine>` 是您希望 NetScaler CPX 实例启动的处理引擎数量。对于每个额外的处理引擎，请确保 Docker 主机包含等量的 vCPU 和 GB 内存。例如，如果您要添加 4 个处理引擎，则 Docker 主机必须包含 4 个 vCPU 和 4 GB 内存。

Compose 文件通常遵循类似于以下内容的格式：

```
<container_name>:

  image: <repository>:<tag>

  ports:

    - 9443

    - 22

    - 9080

    - 161/udp

  tty: true

  cap_add:

    - NET_ADMIN

  ulimits:

    core: -1

  volumes:

    - <host_directory_path>:/cpx

  environment:

    - EULA=yes

    - CPX_CORES=<number_processing_engine>

    - CPX_CONFIG= '{"YIELD": "Yes"}'
```

```
CPX_0:
```

```
  image: quay.io/citrix/citrix-k8s-cpx-ingress:13.1-37.38
```

```
ports:

  - 9443

  - 22

  - 9080

  - 161/udp

tty: true

cap_add:

  - NET_ADMIN

ulimits:

  core: -1

volumes:

  - /root/test:/cpx

environment:

  - CPX_CORES=2

  - EULA=yes
```

将 NetScaler CPX 实例添加到 NetScaler Console

如果要管理和监视这些实例，必须将安装在 Docker 主机上的 NetScaler CPX 实例添加到 NetScaler Application Delivery Management (ADM) 软件中。

您可以在首次设置 ADM 时或之后添加实例。

要添加实例，您必须创建实例配置文件并指定每个实例的主机名或 IP 地址，或者指定 IP 地址范围。此实例配置文件包含要添加到 NetScaler Console 的实例的用户名和密码。对于每种实例类型，都有一个默认配置文件可用。例如，`ns-root-profile` 是 NetScaler 实例的默认配置文件。此配置文件由默认的 ADC 管理员凭据定义。如果您更改了实例的默认管理员凭据，则可以为这些实例定义自定义实例配置文件。如果在发现实例后更改了实例的凭据，则必须编辑实例配置文件或创建配置文件，然后重新发现该实例。

先决条件

确保您已完成以下操作：

- 在 Citrix XenServer 上安装了 NetScaler Console 软件。有关详细信息，请参阅 [NetScaler Console Documentation](#)。
- 在 Docker 主机上安装了 NetScaler CPX 实例。

将 NetScaler CPX 实例添加到 ADM：

- 在 Web 浏览器中，键入 **NetScaler Application Delivery Management** 的 IP 地址（例如，`http://192.168.100.1`）。
- 在“用户名”和“密码”字段中，输入管理员凭据。默认管理员凭据为 **nsroot** 和 **nsroot**。
- 导航到“网络 > 实例”>“**NetScaler**”，然后单击“**CPX**”选项卡。
- 单击“**添加**”以在 NetScaler Console 中添加新的 CPX 实例。
- “**添加 NetScaler CPX**”页面打开。输入以下参数的值：
 - 您可以通过提供 CPX 实例的可访问 IP 地址或托管 CPX 实例的 Docker 容器的 IP 地址来添加 CPX 实例。
 - 选择 CPX 实例的配置文件。

- c. 选择要部署实例的站点。
- d. 选择代理。
- e. 作为可选操作，您可以为实例输入键值对。添加键值对可让您以后轻松搜索实例。

6. 单击 **确定**。

注意

如果要重新发现实例，请选择 **网络 > 实例 > NetScaler > CPX**，选择要重新发现的实例，然后从 **选择操作** 下拉列表中单击 **重新发现**。

使用环境变量将 NetScaler CPX 实例添加到 NetScaler Console

您还可以使用环境变量将 NetScaler CPX 实例添加到 NetScaler Console。要添加实例，必须为 NetScaler CPX 实例配置以下环境变量。

- **NS_MGMT_SERVER** - ADM IP 地址/FQDN
- **HOST** - 节点 IP 地址
- **NS_HTTP_PORT** - 节点上映射的 HTTP 端口
- **NS_HTTPS_PORT** - 节点上映射的 HTTPS 端口

- `NS_SSH_PORT` - 节点上映射的 SSH 端口
- `NS_SNMP_PORT` - 节点上映射的 SNMP 端口
- `NS_ROUTABLE` - (NetScaler CPX pod IP 地址无法从外部路由。)
- `NS_MGMT_USER` – ADM 用户名
- `NS_MGMT_PASS` – ADM 密码

以下是将 NetScaler CPX 实例添加到 NetScaler Console 的 `docker run` 命令示例。

```
docker run -dt --privileged=true -p 9080:9080 -p 9443:9443 -p 9022:22 -p 9161:161 -e EULA=yes -e NS_MGMT_SERVER=abc-mgmt-server.com -e HOST=10.1.1.1 -e NS_HTTP_PORT=9080 -e NS_HTTPS_PORT=9443 -e NS_SSH_PORT=9022 -e NS_SNMP_PORT=9161 -e NS_ROUTABLE=0 --ulimit core=-1 -name test cpx:latest
```

使用 Kubernetes ConfigMaps 将 NetScaler CPX 实例添加到 NetScaler Console

NetScaler CPX 支持通过 Kubernetes ConfigMaps 使用卷挂载文件向 NetScaler Console 注册。要启用此注册方式，NetScaler CPX 需要一些环境变量，这些变量需要与通过 ConfigMaps 和 Secrets 挂载的一些文件一起指定。

以下是所需的环境变量及其说明：

- `NS_HTTP_PORT` - 指定节点上映射的 HTTP 端口。
- `NS_HTTPS_PORT` - 指定节点上映射的 HTTPS 端口。
- `NS_SSH_PORT` - 指定节点上映射的 SSH 端口。
- `NS_SNMP_PORT` - 指定节点上映射的 SNMP 端口。

除了列出的环境变量之外，NetScaler CPX 还需要有关其必须注册的 ADM 代理的信息。此信息包含 ADM 代理的 IP 地址或 FQDN 详细信息以及凭据。NetScaler CPX 从卷挂载文件中获取此信息。包含 IP 地址或 FQDN 的 ConfigMap 作为文件挂载到 NetScaler CPX 实例的文件系统中。包含 ADM 代理凭据的 Kubernetes Secret 也作为文件挂载到 NetScaler CPX 实例的文件系统中。NetScaler CPX 具备注册所需的所有信息后，会尝试向 ADM 代理注册。

以下是 NetScaler CPX YAML 文件片段的示例，其中 ConfigMap 和 Secret 作为文件挂载：

```
...

env:

- name: "EULA"

  value: "yes"

- name: "NS_HTTP_PORT"

  value: "9080"

- name: "NS_HTTPS_PORT"

  value: "9443"

- name: "NS_SSH_PORT"

  value: "22"

- name: "NS_SNMP_PORT"

  value: "161"

- name: "KUBERNETES_TASK_ID"

  value: ""

...

volumeMounts:

  ...

  - mountPath: /var/admininfo/server/

    name: adm-agent-config

  - mountPath: /var/admininfo/credentials/

    name: adm-agent-user

  ...
```

```

volumes:

...

- name: adm-agent-config

  configMap:

    name: adm-agent-config

- name: adm-agent-user

  secret:

    secretName: adm-secret

```

在前面的示例中，使用了名为 `adm-agent-config` 的 ConfigMap 和名为 `adm-agent-user` 的 Secret。以下是创建所需 ConfigMap 和 Secret 的示例。

ConfigMap：ConfigMap 是从名为 `adm_reg_envs` 的文件创建的。该文件需要 ADM 代理的 IP 地址或 FQDN，格式如下：

```
NS_MGMT_SERVER=adm-agent
```

在上述格式中，`adm-agent` 是 NetScaler CPX 实例需要注册到的 ADM 代理的 FQDN。

使用以下命令创建 ConfigMap：

```
kubectl create configmap adm-agent-config --from-file=adm_reg_envs
```

注意： 文件名必须包含 `adm_reg_envs` 变量，并且必须挂载到路径：`/var/admininfo/server/`。

Secret：使用以下命令创建 Kubernetes secret。在以下命令中，`user123` 是 ADM 代理的用户名，`pass123` 是密码。

```
kubectl create secret generic adm-secret --from-
literal=NS_MGMT_USER=user123 --from-literal=NS_MGMT_PASS=pass123
```

即使在集群中部署 ADM 代理之前，也可以在 Kubernetes 集群中部署 NetScaler CPX 实例，并使用所需的环境变量和卷挂载文件。如果在部署 ADM 代理之前部署 NetScaler CPX 实例，NetScaler CPX 将持续尝试注册，直到 ADM 代理部署完成。一旦 ADM 代理部署完成，NetScaler CPX 实例将使用通过环境变量和卷挂载文件提供的配置数据向 ADM 代理注册。这有助于避免重新部署包含配置信息的 NetScaler CPX。

已注册到 ADM 代理的 NetScaler CPX 实例可以在配置更改后动态地将注册更改为另一个 ADM 代理。为此，您可以更新已部署的 NetScaler CPX 的 ConfigMap 和 Secret 中的配置信息。您必须使用新 ADM 代理的 IP 地址或 FQDN 更新创建 ConfigMap 的文件，然后删除旧的 ConfigMap，并创建一个新的 ConfigMap。同样，必须删除现有的 secret，并使用新 ADM 代理的凭据创建一个新的 secret。

NetScaler CPX 许可证聚合器

目前，NetScaler CPX 从 NetScaler Console 服务器获取许可证。在 Kubernetes 环境中，NetScaler CPX 可以动态启动或关闭。如果 NetScaler CPX 意外关闭，NetScaler Console 服务器需要几分钟才能回收许可证。NetScaler Console 服务器必须能够在 NetScaler CPX 关闭时立即回收此类许可证，以便可以将相同的许可证分配给另一个即将启动的 NetScaler CPX。此外，如果 NetScaler Console 服务器因任何原因无法访问，则无法为集群中的新 NetScaler CPX 授予许可证。

NetScaler CPX 许可证聚合器是 NetScaler 提供的一项 Kubernetes 服务。此服务充当 Kubernetes 集群内 NetScaler CPX 许可的本地提供商。部署在 Kubernetes 集群中的 NetScaler CPX 许可证聚合器服务可以充当 NetScaler CPX 和 ADM 许可服务器之间的中间人，并跟踪 NetScaler CPX 和已分配的许可证。借助 NetScaler CPX 许可证聚合器服务，NetScaler Console 服务器可以在 NetScaler CPX 关闭时立即回收许可证。

在 Kubernetes 集群中，NetScaler CPX 许可证聚合器服务支持 NetScaler CPX 作为 sidecar 和独立部署。

注意：

使用 NetScaler CPX 许可证聚合器进行许可需要 NetScaler CPX 13.1-30.x 或更高版本。
NetScaler CPX 许可证聚合器不支持为旧版本的 NetScaler CPX 授予许可证。

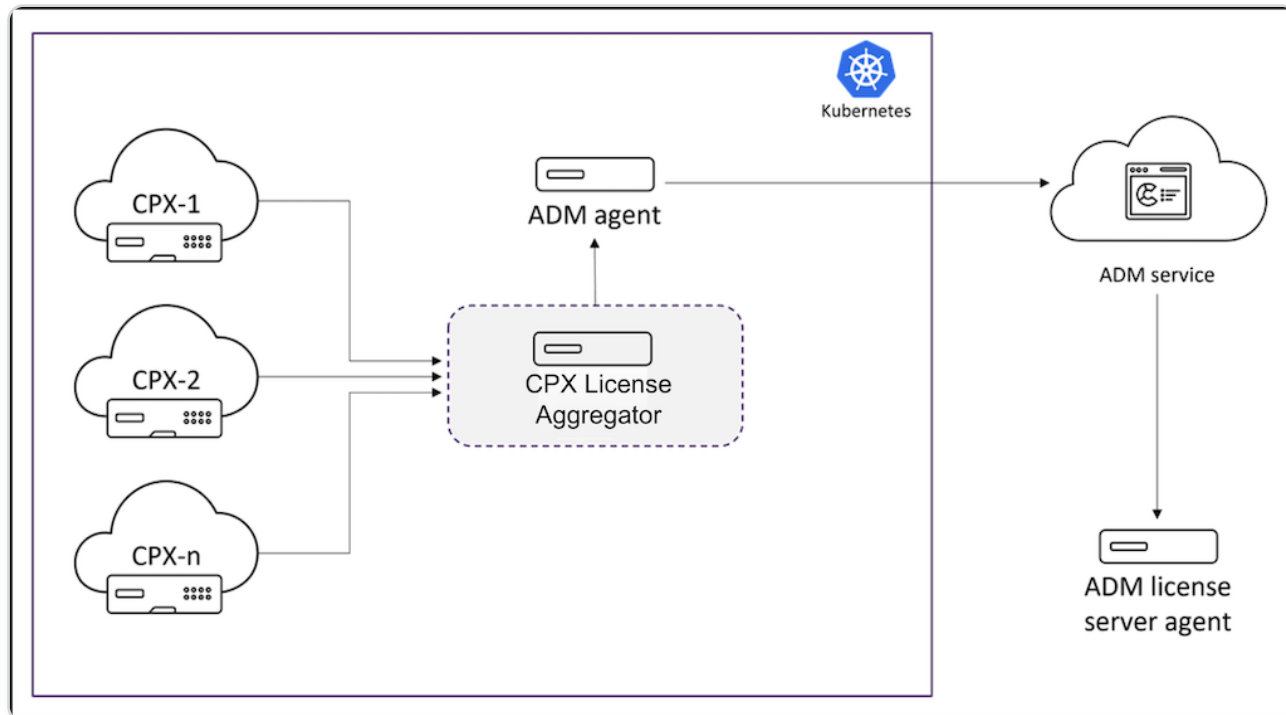
NetScaler CPX 许可证聚合器的主要优势

以下是使用 NetScaler CPX 许可证聚合器的主要优势：

- **可扩展性：**NetScaler Console 许可服务器最多只能支持 10000 个 NetScaler CPX 部署。随着 NetScaler CPX 许可证聚合器服务的引入，每个 Kubernetes 集群都可以作为 NetScaler Console 许可服务器的单个客户端。因此，您可以使用单个 NetScaler Console 许可服务器扩展许多 NetScaler CPX。
- **资源优化：**NetScaler CPX 许可证聚合器服务还支持集群范围的许可功能，并且可以根据需要从 NetScaler Console 服务器签出许可证。NetScaler CPX 许可证聚合器可以将许可证返回给 NetScaler Console 服务器。NetScaler CPX 许可证聚合器可以处理 NetScaler CPX 的非正常终止，并在配置的等待期后从这些 NetScaler CPX 回收许可证。

部署拓扑

下图显示了 Kubernetes 集群中的 NetScaler CPX 许可证聚合器部署。



在此图中：

- CPX 表示 NetScaler CPX
- CPX License Aggregator 表示 NetScaler CPX 许可证聚合器

在此示例部署中，NetScaler CPX License Aggregator 服务与 NetScaler CPX 和 NetScaler 代理一起部署在 Kubernetes 集群中。NetScaler CPX License Aggregator 服务充当 NetScaler CPX 和 NetScaler 代理之间的中介，并监控集群中的所有 NetScaler CPX 并为其管理许可证。

使用 Helm 图表 部署 NetScaler CPX 许可证聚合器

先决条件

适用以下先决条件：

- 您需要 Kubernetes 1.16 版及更高版本。
- 您需要 Helm 3.x 版或更高版本。
- 您必须获取拥有 NetScaler CPX 许可证的许可证服务器的 IP 地址。
- 您必须为 NetScaler CPX License Aggregator 中的 Redis 数据库提供密码。您可以使用 Kubernetes secret 提供数据库密码，并可以使用以下命令创建 secret：

```
kubectl create secret generic dbsecret --from-  
literal=password=&lt;custom-password>
```

使用 Helm chart 进行部署

根据 NetScaler CPX 许可证的类型，执行以下步骤以使用 Helm chart 部署 NetScaler CPX License Aggregator。有关不同类型的 NetScaler CPX 许可证的更多信息，请参阅 [NetScaler CPX 许可](#)。

安装 Helm chart

使用以下命令添加 NetScaler CPX License Aggregator Helm chart 存储库：

```
helm repo add Citrix https://citrix.github.io/citrix-helm-charts/
```

安装 NetScaler CPX License Aggregator 以管理带宽池许可证

根据您拥有的 NetScaler CPX 池许可证的类型，使用以下命令之一。在这些命令中，`my-release` 用作发布名称。

注意：

默认情况下，Helm chart 会安装推荐的 RBAC 角色和角色绑定。

对于白金带宽许可证：

```
helm install my-release citrix/cpx-license-aggregator --set  
licenseServer.address=&lt;License-Server-IP-or-  
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-  
password>,licenseAggregator.username=&lt;unique-ID-for-  
CLA>,licenseInfo.instanceQuantum=&lt;QUANTUM>,licenseInfo.instanceLowWater  
mark=&lt;LOW WATERMARK>,licenseInfo.bandwidthPlatinumQuantum=&lt;QUANTUM-  
in-Mbps>,licenseInfo.bandwidthPlatinumLowWatermark=&lt;LOW WATERMARK-in-  
Mbps>
```

对于企业带宽版本：

```
helm install my-release citrix/cpx-license-aggregator --set  
licenseServer.address=&lt;License-Server-IP-or-  
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-  
password>,licenseAggregator.username=&lt;unique-ID-for-
```

```
CLA>,licenseInfo.instanceQuantum=<QUANTUM>,licenseInfo.instanceLowWater
mark=<LOW
WATERMARK>,licenseInfo.bandwidthEnterpriseQuantum=<QUANTUM-in-
Mbps>,licenseInfo.bandwidthEnterpriseLowWatermark=<LOW WATERMARK-in-
Mbps>
```

对于标准带宽版本：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=<License-Server-IP-or-
FQDN>,redis.secretName=<Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=<unique-ID-for-
CLA>,licenseInfo.instanceQuantum=<QUANTUM>,licenseInfo.instanceLowWater
mark=<LOW WATERMARK>,licenseInfo.bandwidthStandardQuantum=<QUANTUM-
in-Mbps>,licenseInfo.bandwidthStandardLowWatermark=<LOW WATERMARK-in-
Mbps>
```

这些命令使用默认配置在 Kubernetes 集群上部署 NetScaler CPX 许可证聚合器。您可以在安装时配置参数。有关更多信息，请参阅 [Helm chart GitHub 存储库](#) 中的 **NetScaler CPX 许可证聚合器配置** 部分，其中列出了您可以在安装期间配置的强制和可选参数。

安装 NetScaler CPX 许可证聚合器以管理 vCPU 许可证

根据您拥有的 NetScaler CPX vCPU 许可证类型，使用以下命令之一。在这些命令中，`my-release` 用作发布名称。

注意：默认情况下，Helm chart 会安装推荐的 RBAC 角色和角色绑定。

对于白金 vCPU 版本：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=<License-Server-IP-or-
FQDN>,redis.secretName=<Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=<unique-ID-for-
CLA>,licenseInfo.vcpuPlatinumQuantum=<QUANTUM>,licenseInfo.vcpuPlatinum
LowWatermark=<LOW WATERMARK>
```

对于企业 vCPU 版本：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.vcpuEnterpriseQuantum=&lt;QUANTUM>,licenseInfo.vcpuEnterp
riseLowWatermark=&lt;LOW WATERMARK>
```

对于标准 vCPU 版本：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.vcpuStandardQuantum=&lt;QUANTUM>,licenseInfo.vcpuStandard
LowWatermark=&lt;LOW WATERMARK>
```

安装 NetScaler CPX 许可证聚合器以管理多个许可证

如果您需要 NetScaler CPX 许可证聚合器来管理多种类型的许可证，则必须在 Helm 命令中指定这些许可证的相关参数。

例如：

要部署用于 **pooled platinum bandwidth edition** 和 **vCPU platinum edition** 许可证的 NetScaler CPX 许可证聚合器：

```
helm install demo citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.instanceQuantum=&lt;QUANTUM>,licenseInfo.instanceLowWater
mark=&lt;LOW WATERMARK>,licenseInfo.bandwidthPlatinumQuantum=&lt;QUANTUM-
in-Mbps>,licenseInfo.bandwidthPlatinumLowWatermark=&lt;LOW WATERMARK-in-
Mbps>,licenseInfo.vcpuPlatinumQuantum=&lt;QUANTUM>,licenseInfo.vcpuPlatinu
mLowWatermark=LOW WATERMARK>
```

配置 NetScaler CPX 以从 NetScaler CPX 许可证聚合器获取许可证

当您使用 NetScaler CPX 许可证聚合器为 NetScaler CPX 授权时，需要在 NetScaler CPX 部署 YAML 中提供环境变量 **CLA**。

必须在此环境变量中提供可用于访问 NetScaler CPX 许可证聚合器的 `ipaddress` 或 `domainname`，如下所示：

```
env:

- name: "CLA"

  value: "192.0.2.2"
```

或者

```
env:

- name: "CLA"

  value: "local-cla.org"
```

您还必须在 NetScaler CPX YAML 中提供以下环境变量。

- `POD_NAME`：指定 Pod 的名称。Pod 的名称作为环境变量公开给 NetScaler CPX。
- `POD_NAMESPACE`：指定 Pod 的命名空间。Pod 的命名空间作为环境变量公开给 NetScaler CPX。
- `Bandwidth`：指定分配给 NetScaler CPX 的带宽（以 Mbps 为单位）。
- `Edition`：指定许可证版本。支持的值包括 Standard、Platinum 和 Enterprise。
- `CPX_CORES`：指定要为 NetScaler CPX 运行的核心数量。

有关不同的 NetScaler CPX 许可选项的更多信息，请参阅 [NetScaler CPX 许可](#)。

以下显示了包含这些环境变量的示例配置：

```
- name: POD_NAME

valueFrom:

  fieldRef:

    apiVersion: v1

    fieldPath: metadata.name
```

```
- name: POD_NAMESPACE

valueFrom:

  fieldRef:

    apiVersion: v1

    fieldPath: metadata.namespace

- name: "BANDWIDTH"

value: 1000

- name: "CPX_CORES"

value: 1

- name: "EDITION"

value: PLATINUM
```

您还需要将以下标签添加到 NetScaler CPX YAML 中：

```
labels:

  adc: citrix®
```

有关 NetScaler CPX 许可证聚合器的示例部署，请参阅 [NetScaler CPX 许可证聚合器：示例部署](#)。

配置 NetScaler CPX

您可以通过 Linux Docker 主机访问 CLI 提示符，或使用 NetScaler NITRO API 来配置 NetScaler CPX 实例。

使用命令行界面配置 NetScaler CPX 实例

访问 Docker 主机并登录到实例的 SSH 提示符，如以下图所示。登录 NetScaler CPX 实例的默认管理员凭据是 root/linux。

```
root@ubuntu:~# ssh -p 32777 root@127.0.0.1
root@127.0.0.1's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Dec 15 02:45:42 2015 from 172.17.0.1
root@10:~#
```

键入以下命令以使用实例的命令行提示符运行 CLI 命令：`cli_script.sh "<command>"`

示例：

```
root@10:~# cli_script.sh "show ip"
exec: show ip
      Ippaddress      Traffic Domain  Type
      -----
1)      172.17.0.4      0              NetScaler IP|VIP
2)      192.0.0.1      0              SNIP
```

要注销实例提示符，请键入 `logout`。

NetScaler CPX 中支持使用非默认密码

NetScaler CPX 支持为 root 帐户使用非默认密码，即 `nsroot`。部署 NetScaler CPX 后，将生成一个默认密码并分配给用户。此默认密码也会为 SSH 用户 `root` 和 `nsroot` 更新。您可以手动更改此默认密码。您还可以手动重置 `root` 和 `nsroot` 用户帐户的默认 SSH 密码。Citrix 建议手动更改此密码以维护系统的安全性。

重置密码后，新密码将用于 NITRO API 通信和 `cli_script.sh` 执行。默认的 root 帐户密码以纯文本形式存储在 NetScaler CPX 文件系统中的 `/var/deviceinfo/random_id` 文件中。

使用以下语法运行带凭据的 `cli_script.sh`：

```
cli_script.sh "<command>" ":<user>:<password>"
```

例如，要使用用户 `nsroot` 和密码 `Citrix123` 运行 `cli_script.sh` 以显示 IP 地址，请使用以下命令：

```
cli_script.sh "show ns ip" ":nsroot:Citrix123"
```

使用 NITRO API 配置 NetScaler CPX 实例

您可以使用 NetScaler NITRO API 配置 NetScaler CPX 实例。

要使用 Nitro API 配置 NetScaler CPX 实例，请在 Web 浏览器中键入：

```
http://<host_IP_address>:<port>/nitro/v1/config/<resource-type>
```

要使用 Nitro API 检索统计信息，请在 Web 浏览器中键入：

```
http://<host\_IP\_address>:<port>/nitro/v1/stat/<resource-type>
```

有关使用 NITRO API 的更多信息，请参阅 [REST Web Services](#)。对于 NetScaler CPX，请在提及 `netScaler-ip-address` 的地方使用 `CPX IP address:port`。

使用作业配置 NetScaler CPX 实例

您可以通过在 NetScaler Console 中创建和运行作业来配置 NetScaler CPX 实例。您可以使用配置模板中的配置，提取其他设备上可用的配置，并使用保存在文本文件中的配置。您还可以记录使用其他实例的配置实用程序完成的配置。然后，NetScaler Console 会显示相应的 CLI 命令，供您在 NetScaler CPX 实例上使用。选择配置后，您必须选择要加载配置的 **NetScaler CPX 实例**，指定变量值，然后运行作业。

要使用作业配置 NetScaler CPX 实例：

1. 使用管理凭据登录到 NetScaler Console。
2. 导航到 **Networks \> Configuration Jobs**，然后单击 **Create Job**。
3. 指定所需值，并选择配置源。您还可以键入要运行的命令。

Select Configuration | **Select Instances** | **Specify Variable Values**

Job Name*
cpx-single-host

Instance Type*
NetScaler

Configuration Editor

Configuration Source
Configuration Template

Drag and drop the template to the Commands field in the right pane. You can also edit the configuration and save the template with a different name

LBVariablesTemplate

		New
1	SSH	add service db1 172.17.0.10 HTTP 80
2	SSH	add service db2 172.17.0.11 HTTP 80
3	SSH	add lb vserver cpx-vip HTTP 172.17.0.4 81
4	SSH	bind lb vserver cpx-vip db1
5	SSH	bind lb vserver cpx-vip db2

4. 选择要运行配置的 NetScaler CPX 实例，然后单击 **Next**。

Select Configuration | **Select Instances** | **Specify Variable Values**

Select the target instances on which you want to run the configuration.

Add Instances | Delete

	IP Address	Name
☐	172.17.0.150	10.102.31.190

Cancel | Back | **Next** | Save and Exit

5. 指定执行设置，然后单击“完成”以在 NetScaler CPX 实例上运行命令。如果要保存配置并稍后运行，请单击“保存并退出”。

 Select Configuration

 Select Instances

 Specify Variable Values

 Execute

You can either run the commands now or schedule to run the commands at a later time.

On Command Failure*

Ignore error and continue

Execution Mode*

Now

Execution Settings

☒ Execute in Sequence

☐ Execute in Parallel

Receive Execution Report Through

☐ Email

Cancel

← Back

Finish

Save and Exit

在 NetScaler CPX 实例上配置 AppFlow

您可以在 NetScaler CPX 实例上配置 AppFlow 功能，以收集网页性能数据、流和用户会话级别信息以及应用程序性能监控和分析所需的数据库信息。这些数据记录将发送到 NetScaler Console，您可以在其中查看所有应用程序的实时和历史报告。

要配置 AppFlow，首先必须启用 AppFlow 功能。然后，指定流记录发送到的收集器。之后，定义操作，操作是配置的收集器集。然后配置一个或多个策略，并将一个操作与每个策略关联。该策略指示 NetScaler CPX 选择其流记录发送到关联操作的请求。最后，将每个策略全局绑定或绑定到特定的虚拟服务器以使其生效。

您可以进一步设置 AppFlow 参数，以指定模板刷新闻隔并启用 `httpURL`、`httpCookie` 和 `httpReferer` 信息的导出。在每个收集器上，您必须将 NetScaler CPX IP 地址指定为导出器的地址。

配置实用程序提供了帮助用户定义策略和操作的工具。它精确地确定 NetScaler CPX 如何将特定流的导出记录发送到一组收集器（操作）。命令行界面为喜欢命令行的经验用户提供了一组相应的基于 CLI 的命令。

在监控记录之前，您必须将 NetScaler CPX 实例添加到 NetScaler Console。有关将 NetScaler CPX 实例添加到 NetScaler Console 的更多信息，请参阅 [使用 NetScaler Console 安装 NetScaler CPX 实例](#)。

启用 AppFlow®

要使用 AppFlow 功能，您必须首先启用它。

使用命令行界面启用 AppFlow 功能：

运行以下命令：

```
enable ns feature AppFlow

enable ns mode ulfd
```

指定收集器

收集器接收 NetScaler 生成的 AppFlow 记录。要发送 AppFlow 记录，您必须指定至少一个收集器。默认情况下，收集器在 UDP 端口 4739 上侦听 IPFIX 消息。您可以在配置收集器时更改默认端口。

使用命令行界面指定收集器：

使用以下命令添加收集器：

```
add appflow collector <name> -IPAddress <ipaddress> -port  
<port_number> -netprofile <netprofile_name> -Transport Logstream
```

要验证配置，请使用以下命令：

```
show appflow collector <name>
```

要使用命令行界面指定多个收集器：

使用以下命令添加相同数据并将其发送到多个收集器：

```
add appflow collector <collector1> -IPAddress <IP> -Transport  
Logstream  
  
add appflow collector <collector2> -IPAddress <IP> -Transport  
Logstream  
  
add appflow action <action> -collectors <collector1> <collector2>  
-Transport Logstream  
  
add appflow policy <policy> true <action> -Transport Logstream  
  
bind lbvserver <lbvserver> -policy <policy> -priority <priority>  
-Transport Logstream
```

配置 AppFlow 操作

AppFlow 操作是一个设置的收集器，如果关联的 AppFlow 策略匹配，流记录将发送到该收集器。

使用以下命令配置 AppFlow 操作：

```
add appflow action <name> --collectors <string> ... [-  
clientSideMeasurements (Enabled|Disabled) ] [-comment <string>]
```

要验证配置，请使用以下命令：

```
show appflow action
```

配置 AppFlow 策略

配置 AppFlow 操作后，接下来必须配置 AppFlow 策略。AppFlow 策略基于规则，该规则由一个或多个表达式组成。

要使用命令行界面配置 AppFlow 策略：

在命令提示符下，键入以下命令以添加 AppFlow 策略并验证配置：

```
add appflow policy <name> <rule> <action>
```

```
show appflow policy <name>
```

绑定 AppFlow 策略

要使策略生效，您必须全局绑定它，以便它适用于流经 NetScaler CPX 的所有流量。

要使用命令行界面全局绑定 AppFlow 策略：

使用以下命令全局绑定 AppFlow 策略：

```
bind appflow global <policyName> <priority>  
[<gotoPriorityExpression [-type <type>] [-invoke (<labelType>  
<labelName>)]
```

使用以下命令验证配置：

```
show appflow global
```

使用配置文件配置 NetScaler CPX

您可以使用静态配置文件在部署 NetScaler CPX 实例时配置 NetScaler CPX，而不是使用命令行界面 (`cli_script.sh`)、NITRO API 或 NetScaler Console 配置作业来配置 NetScaler CPX。

您可以在部署 NetScaler CPX 容器时提供静态配置文件作为输入文件。在 NetScaler CPX 容器启动期间，容器将根据静态配置文件中指定的配置进行配置。此配置包括 NetScaler 特定的配置和您可以在 NetScaler CPX 容器上动态运行的 bash shell 命令。

静态配置文件的结构

如前所述，部署 NetScaler CPX 时，它会根据静态配置文件中指定的配置进行配置。

静态配置文件是一个 `.conf` 文件，包含两个标签：`#NetScaler Commands` 和 `#Shell Commands`。在 `#NetScaler Commands` 标签下，您必须添加所有 NetScaler 命令以在 NetScaler CPX 上配置 NetScaler 特定配置。在 `#Shell Commands` 标签下，您必须添加要在 NetScaler CPX 上运行的 shell 命令。

在 NetScaler CPX 容器部署期间，NetScaler 命令和 shell 命令将按照配置文件中指定的顺序在容器上运行。

重要提示：

- 标签可以在配置文件中重复多次。
- 标签不区分大小写。
- 配置文件必须作为 `cpx.conf` 文件存在于容器文件系统目录中的 `/etc` 目录中。
- 配置文件还可以包含注释。您必须在注释前添加“#”字符。
- 如果使用配置文件部署 NetScaler CPX 容器时出现故障，故障将记录在容器中的 `ns.log` 文件中。
- 当您重新启动 NetScaler CPX 容器时，配置文件将重新应用于容器。

```
#NetScaler Commands
```

```
add lb vserver v1 http 1.1.1.1 80
```

```
add service s1 2.2.2.2 http 80
```

```
bind lb vserver v1 s1
```

```
#Shell Commands
```

```
touch /etc/a.txt
```

```
echo "this is a" > /etc/a.txt
```

```
#NetScaler Commands
```

```
add lb vserver v2 http
```

```
#Shell Commands
```

```
echo "this is a 1" >> /etc/a.txt
```

```
#NetScaler Commands
```

```
add lb vserver v3 http
```

```
#This is a test configuration file
```

要安装 NetScaler CPX 容器并根据配置文件动态配置 NetScaler CPX 容器，请使用 `docker run` 命令中的 `-v` 选项挂载静态配置文件：

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -v /tmp/  
cpx.conf:/etc/cpx.conf --name mycpx store/citrix/citrixadccpx:13.0-x.x
```

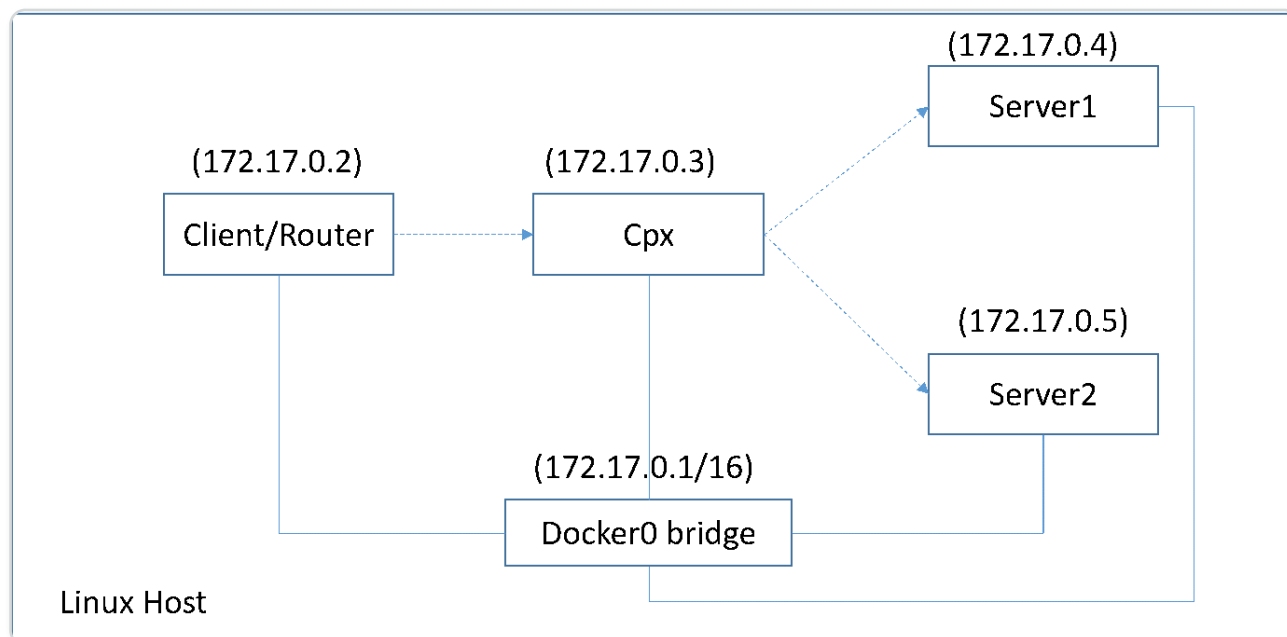
NetScaler CPX 中的动态路由支持

NetScaler CPX 支持 BGP 动态路由协议。动态路由协议的主要目标是根据绑定到虚拟服务器的服务的运行状况来通告虚拟服务器的 IP 地址。它有助于上游路由器在通往拓扑分布的虚拟服务器的多个路由中选择最佳路由。

有关 NetScaler CPX 中非默认密码的信息，请参阅 [配置 NetScaler CPX](#) 文档中的 `Support for using a non-default password in NetScaler CPX` 部分。

在单个主机网络中，客户端、服务器和 NetScaler CPX 实例作为容器部署在同一个 Docker 主机上。所有容器都通过 docker0 网桥连接。在此环境中，NetScaler CPX 实例充当在同一 Docker 主机上作为容器配置的应用程序的代理。有关 NetScaler CPX 主机网络模式部署的信息，请参阅 [主机网络模式](#)。

下图说明了单主机拓扑。



在此拓扑中，虚拟服务器使用 BGP 配置并通告（根据服务的运行状况）到上游网络或路由器。

请执行以下步骤，在采用桥接网络模式的单个 Docker 主机上配置 NetScaler CPX 上的 BGP。

使用 REST API 在 NetScaler CPX 上配置基于 BGP 的路由运行状况注入

1. 使用以下命令从 NetScaler CPX 映像创建容器：

```
docker run -dt --privileged=true -p 22 -p 80 -p 161 -e EULA=yes --  
ulimit core=-1 cpx: <tag>
```

例如：

```
docker run -dt --privileged=true -p 22 -p 80 -p 161 -e EULA=yes --  
ulimit core=-1 cpx:12.1-50.16
```

2. 使用以下命令登录到容器：

```
docker exec -it <container id> bash
```

3. 使用以下命令启用 BGP 功能：

```
cli_script.sh "enable ns feature bgp"
```

4. 使用 `show ns ip` 命令获取 NSIP：

```
cli_script.sh "show ns ip"
```

5. 使用以下命令添加虚拟服务器：

```
cli_script.sh "add lb vserver <vserver_name> http <VIP>  
<PORT>"
```

6. 添加服务并将服务绑定到虚拟服务器。

7. 使用以下命令为 VIP 启用 `hostroute`：

```
cli_script.sh "set ns ip <VIP> -hostroute enabled"
```

从容器中注销，并从主机向端口 9080 上的 NSIP 发送 BGP NITRO 命令。

8. 配置 BGP 路由器：

例如，如果您想配置：

```
router bgp 100

Neighbour 172.17.0.2 remote-as 101

Redistribute kernel
```

将命令指定为：

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/ -X
POST --data 'object={"routerDynamicRouting": {"bgpRouter" :
{"localAS":100, "neighbor": [{ "address": "172.17.0.2", "remoteAS":
101}], "afParams":{"addressFamily": "ipv4", "redistribute":
{"protocol": "kernel"}}}}}'
```

9. 使用以下 NITRO 命令将学习到的 BGP 路由安装到 PE 中：

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/ --data
'object={"params":{"action":"apply"},"routerDynamicRouting":
{"commandstring" : "ns route-install bgp"}}'
```

10. 使用以下 NITRO 命令验证 BGP 邻接状态：

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/
routerDynamicRouting/bgpRouter
```

示例输出：

```
root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/nitro/
v1/config/routerDynamicRouting/bgpRouter

{ "errorcode": 0, "message": "Done", "severity": "NONE",
"routerDynamicRouting":{"bgpRouter":[{"localAS": 100, "routerId":
"172.17.0.3", "afParams": [ { "addressFamily": "ipv4" },
{ "addressFamily": "ipv6" } ], "neighbor": [ { "address": "172.17.0.2",
"remoteAS": 101, "ASOriginationInterval": 15, "advertisementInterval":
30, "holdTimer": 90, "keepaliveTimer": 30, "state": "Connect",
"singlehopBfd": false, "multihopBfd": false, "afParams":
[ { "addressFamily": "ipv4", "activate": true }, { "addressFamily":
"ipv6", "activate": false } ]
```

11. 使用以下命令验证通过 BGP 学习到的路由是否已安装在数据包引擎中：

```
cli_script.sh "show route"
```

12. 使用以下命令保存配置：

```
cli_script.sh "save config"
```

动态路由配置保存在 `/nsconfig/ZebOS.conf` 文件中。

配置 Docker 日志驱动程序

Docker 包含称为“日志驱动程序”的日志机制，可帮助您从正在运行的容器中获取信息。您可以将 NetScaler CPX 容器配置为将其生成的日志转发到 Docker 日志驱动程序。有关 Docker 日志驱动程序的更多信息，请参阅 [配置日志驱动程序](#)。

默认情况下，NetScaler CPX 容器生成的所有日志都存储在 Docker 主机上的 `/cpx/log/ns.log` 文件中。当您使用 `docker run` 命令启动 NetScaler CPX 容器时，可以使用 `--log-driver` 选项将其配置为将所有生成的日志转发到 Docker 日志驱动程序。如果日志驱动程序具有可配置参数，则可以使用 `--log-opt <NAME>=<VALUE>` 选项设置它们。

在以下示例中，NetScaler CPX 容器配置为使用 syslog 作为日志驱动程序转发所有生成的日志。

```
docker run -dt --privileged=true --log-driver syslog --log-opt syslog-address=udp://10.106.102.190:514 -e EULA=yes --ulimit core=-1 --name test store/citrix/cpx:12.1-48.13
```

同样，在以下示例中，NetScaler CPX 容器配置为使用 Splunk 作为日志驱动程序转发所有生成的日志。

```
docker run -dt --privileged=true --log-driver=splunk --log-opt splunk-token=176FCEBF-4CF5-4EDF-91BC-703796522D20 --log-opt splunk-url=https://splunkhost:8088 -e EULA=yes --ulimit core=-1 --name test store/citrix/cpx:12.1-48.13
```

升级 NetScaler CPX 实例

您可以通过关闭 NetScaler CPX 实例，在同一挂载点上安装最新版本，然后删除旧实例来升级它。挂载点是您将主机上的 **/cpx** 目录挂载到的目录。

例如，要将现有 NetScaler CPX 实例的 **/cpx** 目录挂载到主机上的 **/var/cpx** 目录中，挂载点是 **/var/cpx**，NetScaler CPX 挂载目录是 **/cpx**，如下所示：

```
root@ubuntu:~# docker run -dt -e EULA=yes --name mycpx -v /var/cpx:/cpx --ulimit core=-1 cpx:13.0-x.x
```

先决条件

请确保您已具备以下条件：

- 您挂载现有 NetScaler CPX 实例的 **/cpx** 目录的主机目录的详细信息。您可以使用 `docker inspect <containerName>` 命令（其中 `<containerName>` 是 NetScaler CPX 容器的名称）来显示有关主机目录的信息。

该命令的输出提供了容器配置的详细信息，包括卷。在“**Mounts**”条目中，“**Source**”子条目显示了主机上主机目录的位置。

```
"Mounts": [
  {
    "Source": "/var/cpx",
    "Destination": "/cpx",
    "Mode": "",
    "RW": true
  }
],
```

- 下载最新的 NetScaler CPX Docker 镜像文件并加载 NetScaler CPX Docker 镜像。要加载镜像，请导航到您保存 Docker 镜像文件的目录。使用 `docker load -i <image_name>` 命令加载镜像。加载 NetScaler CPX 镜像后，您可以输入 `docker images` 命令来显示有关镜像的信息：

```
root@ubuntu:~# docker load -i cpx-13.0-x.x.gz
```

```
root@ubuntu:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
cpx	13.0-x.x	2e97aadf918b	43 hours ago	414.5 MB

升级 NetScaler CPX 实例

1. 通过输入 `docker stop <containerName>` 命令停止现有 NetScaler CPX 实例，其中 `<containerName>` 是 NetScaler CPX 实例的名称。

```
root@ubuntu:~# docker stop mycpx

mycpx
```

2. 使用 `docker run` 命令，从您已加载到主机上的 NetScaler CPX 镜像部署最新的 NetScaler CPX 实例。确保您在与现有 NetScaler CPX 实例相同的挂载点（例如，`/var/cpx:/cpx`）部署该实例。

```
root@ubuntu:~# docker run -dt -P -e CPX_CORES=1 --name latestcpx --
ulimit core=-1 -e EULA=yes -v /var/cpx:/cpx --cap-add=NET_ADMIN
cpx:13.0-x.x
```

您可以输入 `docker ps` 命令，以便验证部署的 NetScaler CPX 实例是否为最新版本。

```
...
```

```
root@ubuntu:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS		PORTS	NAMES

```
ead12ec4e965    cpx:13.0-x.x    "/bin/sh -c 'bash -C "    5 seconds ago
Up 5 seconds    22/tcp, 80/tcp, 443/tcp, 161/udp    latestcpx
...
```

3. 验证您已部署正确的 NetScaler CPX 实例后，输入 **docker rm** `<containerName>` 命令以删除旧实例。

```
root@ubuntu:~# docker rm mycpx

mycpx
```

在 NetScaler CPX 实例中使用通配符虚拟服务器

当您预置 NetScaler 实例时，Docker 引擎仅为 NetScaler CPX 实例分配一个私有 IP 地址（单个 IP 地址）。NetScaler 实例的三个 IP 功能复用到一个 IP 地址上。此单个 IP 地址使用不同的端口号作为 NSIP、SNIP 和 VIPs。

Docker 引擎分配的单个 IP 地址是动态的。使用单个 IP 地址或 127.0.0.1 IP 地址添加负载均衡 (LB) 或内容交换 (CS) 虚拟服务器。使用 127.0.0.1 创建的虚拟服务器称为通配符虚拟服务器。默认情况下，当您创建通配符虚拟服务器时，NetScaler CPX 会替换通配符虚拟服务器的分配 IP 地址。分配的 IP 地址是 127.0.0.1，它将被替换为 Docker 引擎分配给 NetScaler CPX 实例的 NSIP。

在高可用性 NetScaler CPX 部署中，您可以在主 NetScaler CPX 实例上添加通配符虚拟服务器。节点之间的配置同步会在辅助 NetScaler CPX 实例上配置通配符虚拟服务器。这消除了 Docker 引擎分配给 NetScaler CPX 实例的 NSIP 上配置虚拟服务器的需要。

注意事项：

- 确保您分配给通配符虚拟服务器的端口号未被部署中的任何其他虚拟服务器使用。
- 如果您分配给通配符虚拟服务器的端口号已被内部服务占用，则通配符虚拟服务器添加失败。
- 通配符虚拟服务器不支持 * 字符。

要创建通配符负载均衡虚拟服务器，请在命令提示符下输入以下命令：

```
add lb vserver <name> <serviceType> 127.0.0.1 <port>
```

```
add lb vserver testlbvserver HTTP 127.0.0.1 30000
```

要创建通配符内容交换虚拟服务器，请在命令提示符下输入以下命令：

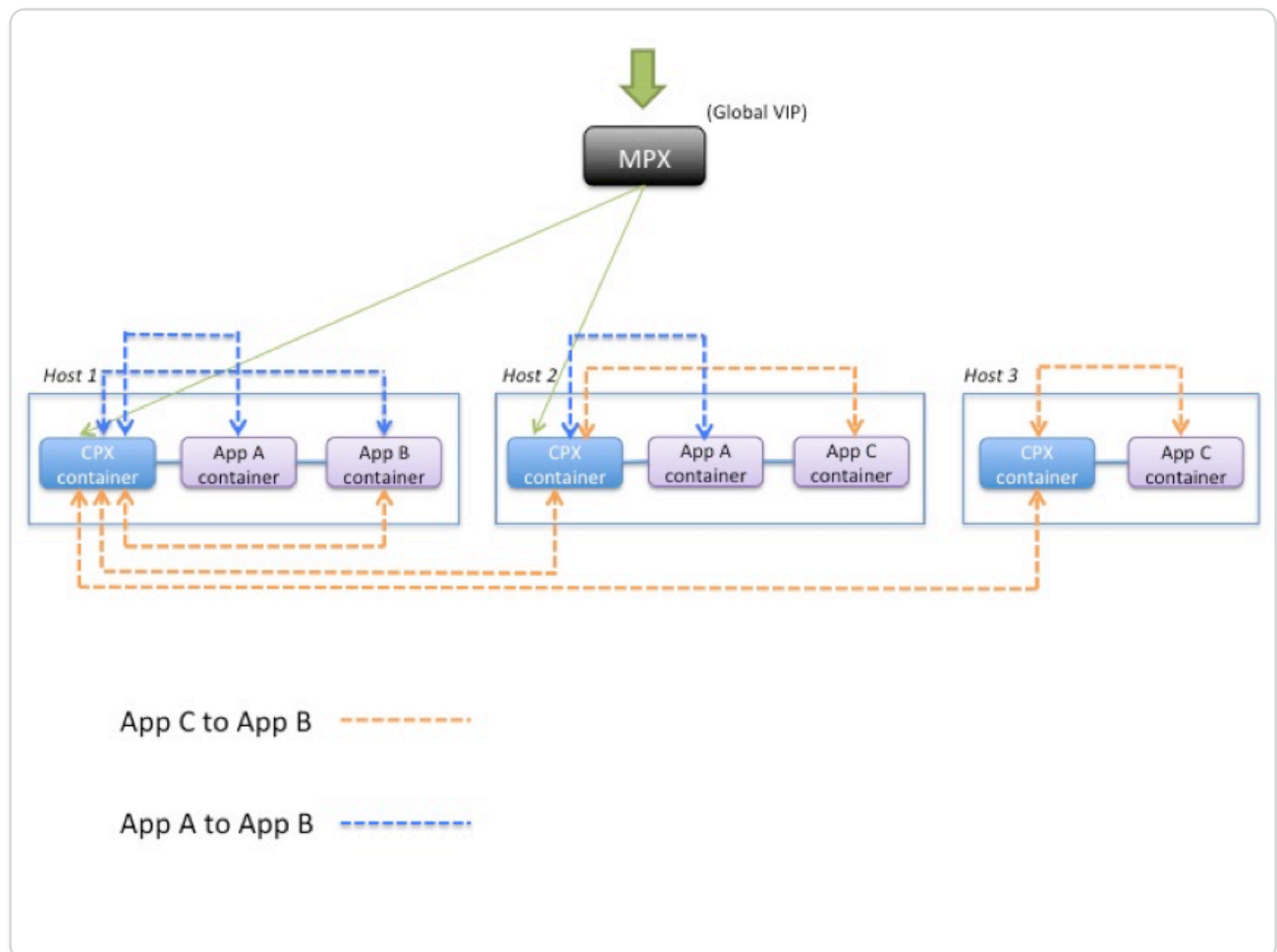
```
add cs vserver <name> <serviceType> 127.0.0.1 <port>
```

```
add cs vserver testcsvserver HTTP 127.0.0.1 30000
```

部署 NetScaler CPX 作为代理以后用东西向流量

在此部署中，NetScaler CPX 实例充当代理，以实现驻留在多个主机上的应用程序容器之间的通信。NetScaler CPX 实例与多个主机中的应用程序一起预置，并提供最短的通信路径。

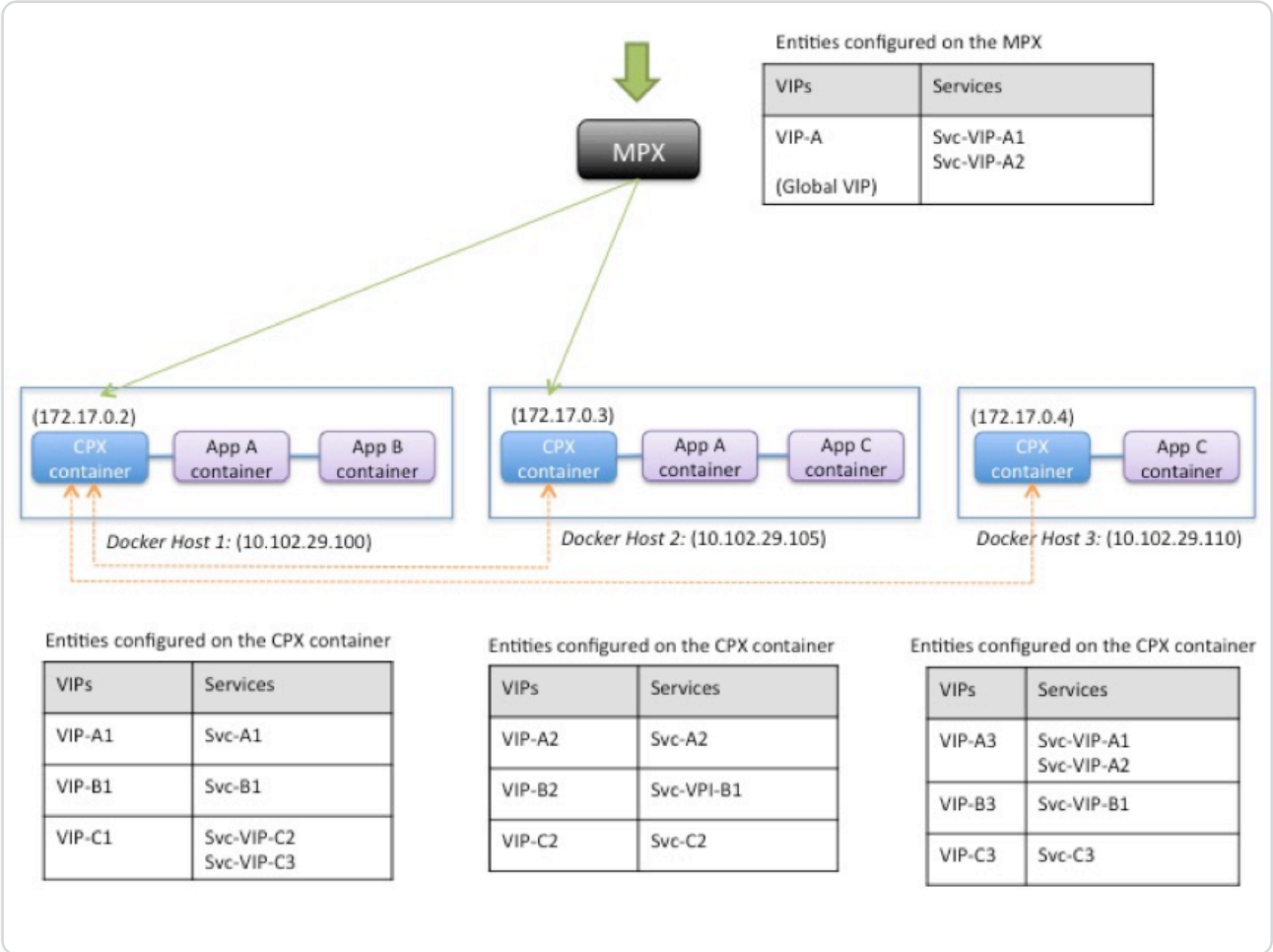
下图说明了通过 NetScaler CPX 实例在两个应用程序之间流动的流量。



此图显示了应用程序 C 和应用程序 B 之间以及应用程序 A 和应用程序 B 之间的流量。当应用程序 C（在任何主机中）向 B 发送请求时，请求首先在与应用程序 C 位于同一主机上的 NetScaler CPX 容器上接收。然后，NetScaler CPX 容器将流量传递到与应用程序 B 位于同一主机上的 NetScaler CPX 容器，然后将流量转发到应用程序 B。当应用程序 A 向应用程序 B 发送请求时，也会遵循类似的流量路径。

在此示例中，还部署了 NetScaler MPX，以允许通过全局 VIP 从 Internet 访问应用程序的流量。来自 NetScaler MPX 的流量在 NetScaler CPX 容器上接收，然后由其将流量分发到各个应用程序容器。

下图说明了此拓扑以及为实现通信而需要设置的配置。



下表列出了在此示例配置中 NetScaler CPX 实例上配置的 IP 地址和端口。

Docker Host 1		Docker Host 2		Docker Host 3	
VIPs	Services Bound to the VIP	VIPs	Services Bound to the VIP	VIPs	Services Bound to the VIP
VIP-A1 172.17.0.2:30000	SVC-A1 10.102.29.100:80	VIP-A2 172.17.0.3:30000	SVC-A2 10.102.29.105:80	VIP-A3 172.17.0.4:30000	SVC-VIP-A1 10.102.29.100:30000
					SVC-VIP-A2 10.102.29.105:30000
VIP-B1 172.17.0.2:30001	SVC-B1 10.102.29.100:90	VIP-B2 172.17.0.3:30001	SVC-VIP-B1 10.102.29.100:30001	VIP-B3 172.17.0.4:30001	SVC-VIP-B1 10.102.29.100:30001
VIP-C1 172.17.0.2:30002	SVC-VIP-C2 10.102.29.105:30002	VIP-C2 172.17.0.3:30002	SVC-C2 10.102.29.105:70	VIP-C3 172.17.0.4:30002	SVC-C3 10.102.29.110:70
	SVC-VIP-C3 10.102.29.110:30002				

要配置此示例场景，请在 Linux shell 提示符下运行以下命令，同时所有三个 Docker 主机上创建 NetScaler CPX 容器：

```
docker run -dt -p 22 -p 80 -p 161/udp -p 30000-30002: 30000-30002 --ulimit core=-1 --privileged=true cpx:6.2
```

请运行以下命令，可以通过使用 NetScaler Console 中的作业功能，或使用 NITRO API。

在 Docker 主机 1 上的 NetScaler CPX 实例上：

```
add lb vserver VIP-A1 HTTP 172.17.0.2 30000

add service svc-A1 10.102.29.100 HTTP 80

bind lb vserver VIP-A1 svc-A1

add lb vserver VIP-B1 HTTP 172.17.0.2 30001

add service svc-B1 10.102.29.100 HTTP 90

bind lb vserver VIP-B1 svc-B1

add lb vserver VIP-C1 HTTP 172.17.0.2 30002

add service svc-VIP-C2 10.102.29.105 HTTP 30002
```

```
add service svc-VIP-C3 10.102.29.110 HTTP 30002

bind lb vserver VIP-C1 svc-VIP-C2

bind lb vserver VIP-C1 svc-VIP-C3
```

在 Docker 主机 2 上的 NetScaler CPX 实例上：

```
add lb vserver VIP-A2 HTTP 172.17.0.3 30000

add service svc-A2 10.102.29.105 HTTP 80

bind lb vserver VIP-A2 svc-A2

add lb vserver VIP-B2 HTTP 172.17.0.3 30001

add service svc-VIP-B1 10.102.29.100 HTTP 30001

bind lb vserver VIP-B2 svc-VIP-B1

add lb vserver VIP-C2 HTTP 172.17.0.3 30002

add service svc-C2 10.102.29.105 HTTP 70

bind lb vserver VIP-C2 svc-C2
```

在 Docker 主机 3 上的 NetScaler CPX 实例上：

```
add lb vserver VIP-A3 HTTP 172.17.0.4 30000

add service svc-VIP-A1 10.102.29.100 HTTP 30000

add service svc-VIP-A2 10.102.29.105 HTTP 30000

bind lb vserver VIP-A3 svc-VIP-A1

bind lb vserver VIP-A3 svc-VIP-A2

add lb vserver VIP-B3 HTTP 172.17.0.4 30001

add service svc-VIP-B1 10.102.29.100 HTTP 30001
```

```
bind lb vserver VIP-B3 svc-VIP-B1

add lb vserver VIP-C3 HTTP 172.17.0.4 30002

add service svc-C3 10.102.29.110 HTTP 70

bind lb vserver VIP-C3 svc-C3
```

在单个主机网络中部署 NetScaler CPX

在单个主机网络中，NetScaler CPX 实例充当同一主机上应用程序容器之间的代理。在此能力下，NetScaler CPX 实例为基于容器的应用程序提供可伸缩性和安全性。此外，它还优化性能并提供对遥测数据的洞察。

在单个主机网络中，客户端、服务器和 NetScaler CPX 实例作为容器部署在同一个 Docker 主机上。所有容器都通过 docker0 网桥连接。

在此环境中，NetScaler CPX 实例充当在同一 Docker 主机上作为容器配置的应用程序的代理。

下图说明了单主机拓扑。

本地化图像(/en-us/cpx/media/cpx-single-host-topology.png)

在此示例中，Web 应用程序容器 (172.17.0.2) 是客户端，两个数据库容器 DB1 (172.17.0.10) 和 DB2 (172.17.0.11) 是服务器。NetScaler CPX 容器 (172.17.0.4) 位于客户端和服务器之间，充当代理。

要使 Web 应用程序通过 NetScaler CPX 与数据库容器通信，您必须首先在 NetScaler CPX 容器上配置两个服务以表示这两个服务器。然后，使用 NetScaler CPX IP 地址和非标准 HTTP 端口（例如 81）配置一个虚拟服务器，因为 NetScaler CPX 将标准 HTTP 端口 80 保留用于 NITRO 通信。

在此拓扑中，您无需配置任何 NAT 规则，因为客户端和服务器位于同一网络上。

要配置此场景，请通过使用 NetScaler Console 中的“作业”功能或通过使用 NITRO API 运行以下命令：

```
add service db1 HTTP 172.17.0.10 80

add service db2 HTTP 172.17.0.11 80

add lb vserver cpx-vip HTTP 172.17.0.4 81

bind lb vserver cpx-vip db1

bind lb vserver cpx-vip db2
```

在多主机网络中部署 NetScaler CPX

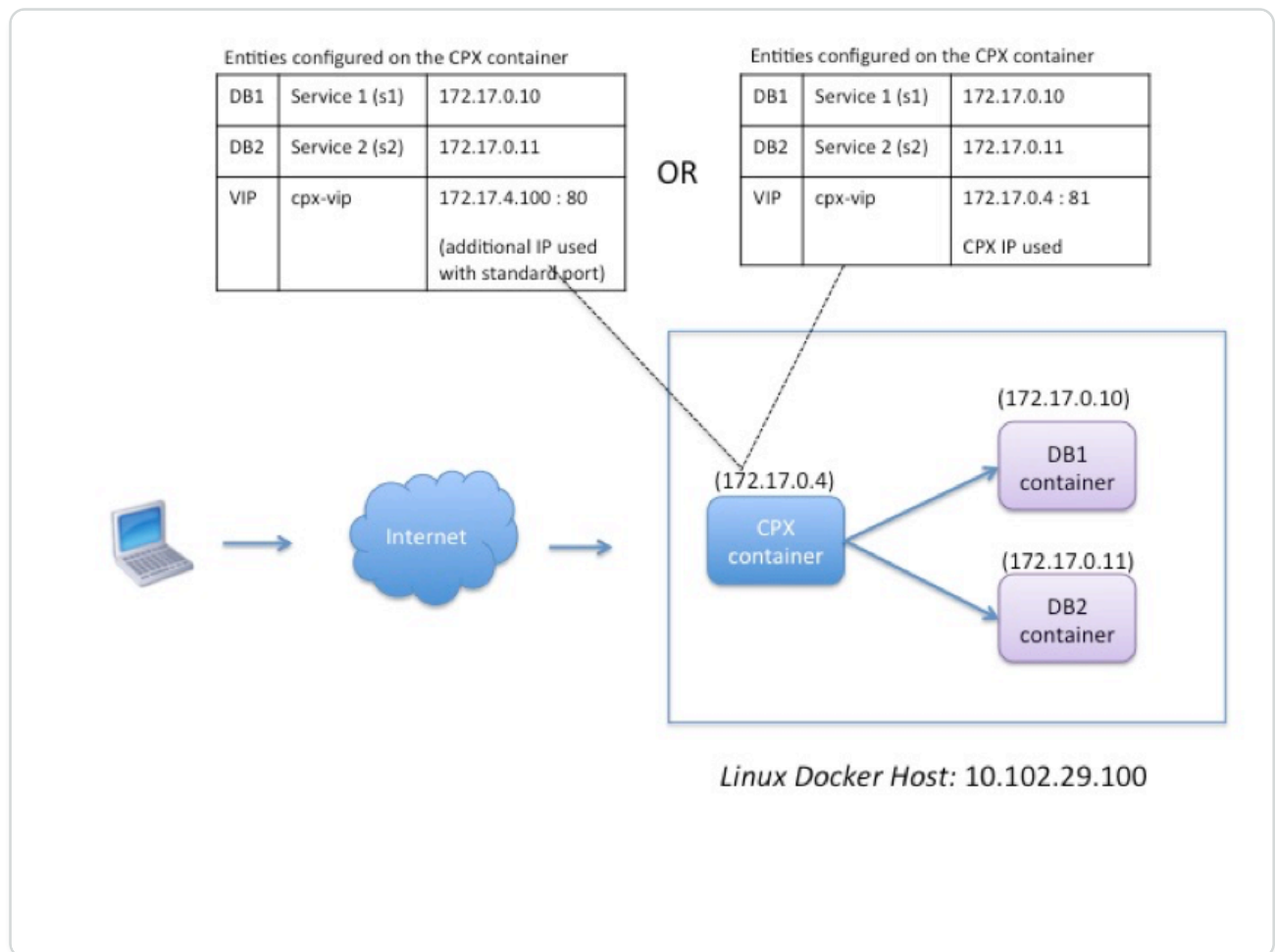
多主机网络中的 NetScaler CPX 实例可以在数据中心的生产部署中进行配置，以提供负载均衡功能。它还可以提供监控功能和分析数据。

在多主机网络中，NetScaler CPX 实例、后端服务器和客户端部署在不同的主机上。您可以在生产部署中使用多主机拓扑，其中 NetScaler CPX 实例对一组基于容器的应用程序和服务器甚至物理服务器进行负载均衡。

拓扑 1：NetScaler CPX 和后端服务器位于同一主机上；客户端位于不同网络上

在此拓扑中，NetScaler CPX 实例和数据库服务器在同一个 Docker 主机上预置，但客户端流量源自网络上的其他位置。此拓扑可用于生产部署，其中 NetScaler CPX 实例对一组基于容器的应用程序或服务器进行负载均衡。

下图说明了此拓扑。



在此示例中，NetScaler CPX 实例 (172.17.0.4) 和两台服务器 DB1 (172.17.0.10) 和 DB2 (172.17.0.11) 在 IP 地址为 10.102.29.100 的同一 Docker 主机上预置。客户端位于网络上的其他位置。

源自 Internet 的客户端请求在 NetScaler CPX 实例上配置的 VIP 上接收，然后 VIP 将请求分发到两台服务器。

您可以使用两种方法来配置此拓扑：

方法 1：为 VIP 使用附加 IP 地址和标准端口

1. 通过使用附加 IP 地址在 NetScaler CPX 容器上配置 VIP。
2. 为 Docker 主机配置附加 IP 地址。
3. 配置 NAT 规则，将 Docker 主机附加 IP 地址上接收到的所有流量转发到 VIP 的附加 IP 地址。
4. 将两台服务器配置为 NetScaler CPX 实例上的服务。
5. 最后，将服务绑定到 VIP。

请注意，在此示例配置中，10.x.x.x 网络表示公共网络。

要配置此示例场景，请使用 NetScaler Console 中的“作业”功能或使用 NITRO API 运行以下命令：

```
add service s1 172.17.0.10 HTTP 80

add service s2 172.17.0.11 HTTP 80

add lb vserver cpx-vip HTTP 172.17.4.100 80

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

通过在 Linux shell 提示符下运行以下命令，为 Docker 主机配置一个额外的公共 IP 地址和 NAT 规则：

```
ip addr add 10.102.29.103/24 dev eth0

iptables -t nat -A PREROUTING -p ip -d 10.102.29.103 -j DNAT --to-destination 172.17.4.100
```

方法 2：使用 NetScaler CPX IP 地址作为 VIP 并配置端口映射：

1. 在 NetScaler CPX 实例上配置 VIP 和两个服务。将非标准端口 81 与 VIP 一起使用。
2. 将服务绑定到 VIP。
3. 配置 NAT 规则，将 Docker 主机端口 50000 上接收到的所有流量转发到 VIP 和端口 81。

要配置此示例场景，请在所有三个 Docker 主机上创建 NetScaler CPX 容器时，在 Linux shell 提示符下运行以下命令：

```
docker run -dt -p 22 -p 80 -p 161/udp -p 50000:81 --ulimit core=-1 --privileged=true cpx:6.2
```

预置 NetScaler CPX 实例后，请使用 NetScaler Console 中的“作业”功能或使用 NITRO API 运行以下命令：

```
add service s1 172.17.0.10 http 80

add service s2 172.17.0.11 http 80

add lb vserver cpx-vip HTTP 172.17.0.4 81
```

```
bind lb vserver cpx-vip s1
```

```
bind lb vserver cpx-vip s2
```

注意：

如果在预置 NetScaler CPX 实例期间未配置端口映射，请通过在 Linux shell 提示符下运行以下命令来配置 NAT 规则：

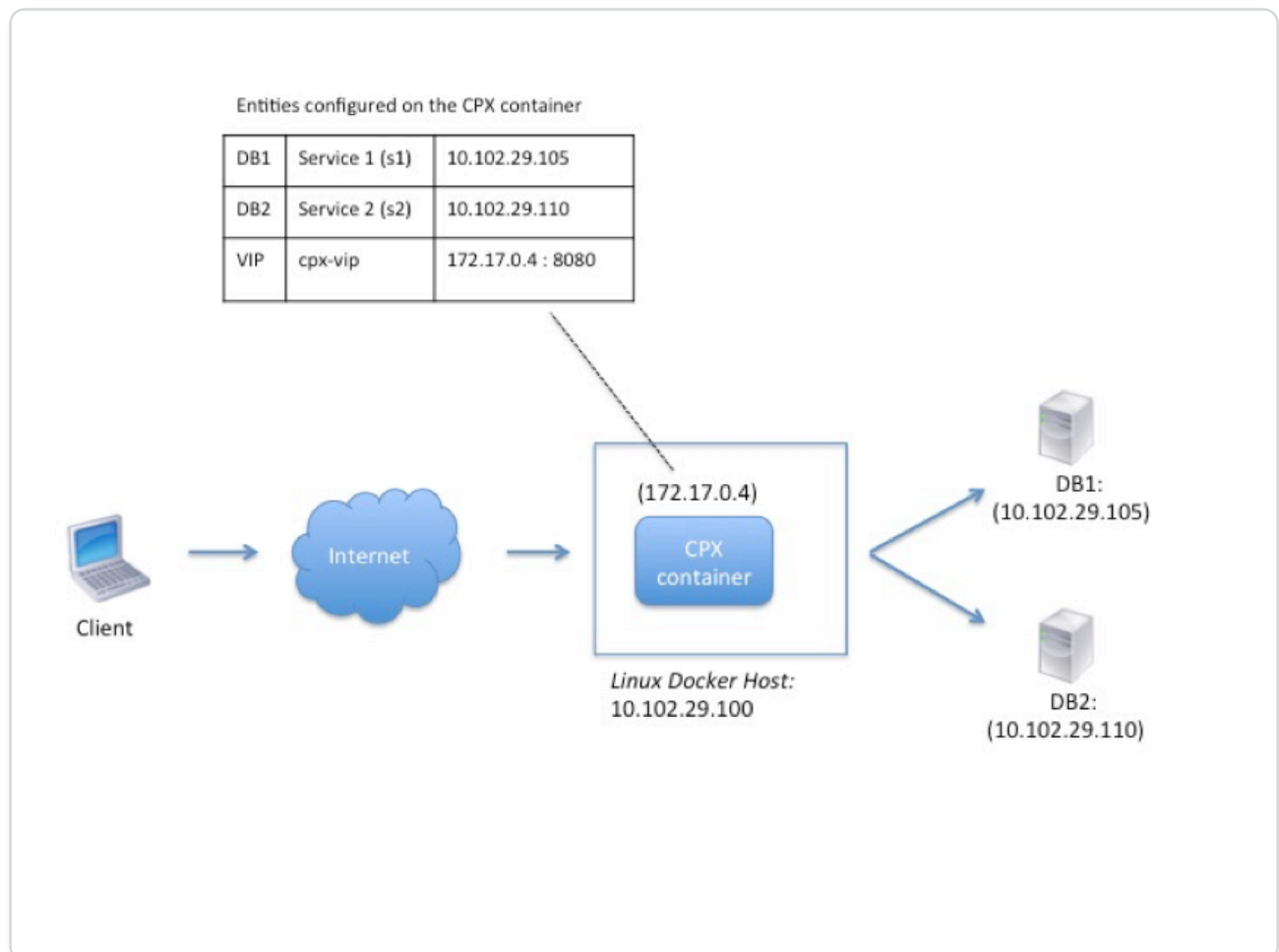
```
iptables -t nat -A PREROUTING -p tcp -m addrtype --dst-type LOCAL -m tcp --dport 50000 -j DNAT --to-destination 172.17.0.4:81
```

拓扑 2：带物理服务器和客户端的 NetScaler CPX

在此拓扑中，只有 NetScaler CPX 实例在 Docker 主机上预置。客户端和服务端不是基于容器的，而是位于网络上的其他位置。

在此环境中，您可以配置 NetScaler CPX 实例以在物理服务器之间进行流量负载均衡。

下图说明了此拓扑。



在此示例中，NetScaler CPX 容器 (172.17.0.4) 位于客户端和充当代理的物理服务器之间。服务器 DB1 (10.102.29.105) 和 DB2 (10.102.29.110) 位于网络上的 Docker 主机外部。客户端请求源自 Internet，并在 NetScaler CPX 上接收，NetScaler CPX 将其分发到两台服务器。

要通过 NetScaler CPX 在客户端和服务器之间启用此通信，您必须首先在创建 NetScaler CPX 容器时配置端口映射。然后，在 NetScaler CPX 容器上配置两个服务以表示两台服务器。最后，使用 NetScaler CPX IP 地址和非标准映射的 HTTP 端口 8080 配置虚拟服务器。

请注意，在示例配置中，10.x.x.x 网络表示公共网络。

要配置此示例场景，请在创建 NetScaler CPX 容器时在 Linux shell 提示符下运行以下命令：

```
docker run -dt -p 22 -p 80 -p 161/udp -p 8080:8080 --ulimit core=-1 --privileged=true cpx:6.2
```

然后，通过使用 NetScaler Console 中的“作业”功能或使用 NITRO API 运行以下命令：

```
add service s1 HTTP 10.102.29.105 80
add service s2 HTTP 10.102.29.110 80
```

```
add lb vserver cpx-vip HTTP 172.17.0.4 8080

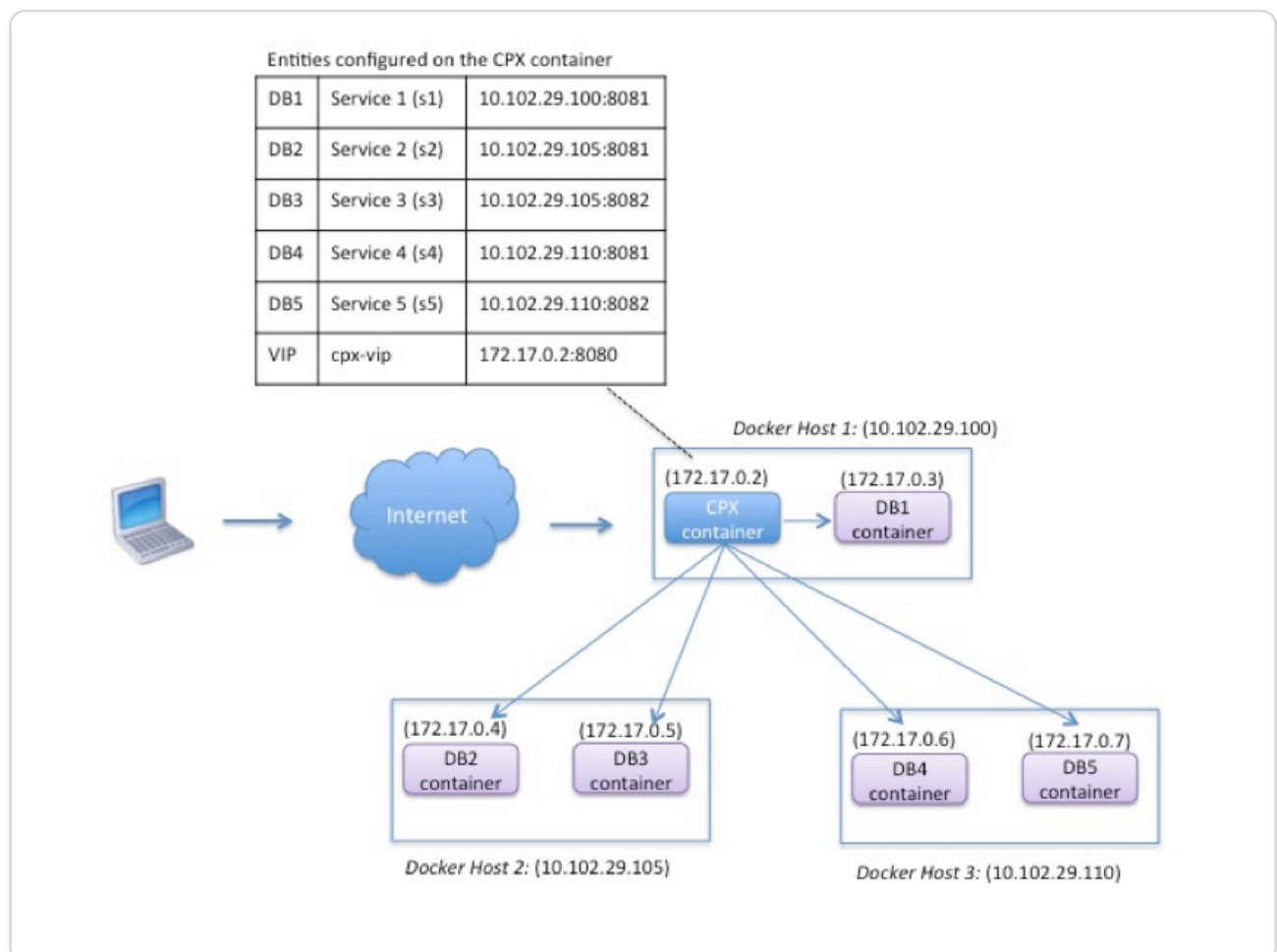
bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

拓扑 3：在不同主机上预置 NetScaler CPX 和服务

在此拓扑中，NetScaler CPX 实例和数据库服务器预置在不同的 Docker 主机中，并且客户端流量源自 Internet。此拓扑可能用于生产部署，其中 NetScaler CPX 实例对一组基于容器的应用程序或服务进行负载均衡。

下图说明了此拓扑。



在此示例中，NetScaler CPX 实例和一台服务器 (DB1) 预置在 IP 地址为 10.102.29.100 的同一 Docker 主机上。其他四台服务器 (DB2、DB3、DB4 和 DB5) 预置在两个不同的 Docker 主机上，分别为 10.102.29.105 和 10.102.29.110。

源自 Internet 的客户端请求在 NetScaler CPX 实例上接收，然后 NetScaler CPX 实例将请求分发到五台服务器。要启用此通信，您必须配置以下内容：

1. 在创建 NetScaler CPX 容器时设置端口映射。在此示例中，这意味着您必须将容器上的端口 8080 转发到主机上的端口 8080。当客户端请求到达主机的端口 8080 时，它会映射到 CPX 容器的端口 8080。
2. 在 NetScaler CPX 实例上将这五台服务器配置为服务。您必须使用相应的 Docker 主机 IP 地址和映射端口的组合来设置这些服务。
3. 在 NetScaler CPX 实例上配置一个 VIP 以接收客户端请求。此 VIP 应由 NetScaler CPX IP 地址和映射到主机端口 8080 的端口 8080 表示。
4. 最后，将服务绑定到 VIP。

请注意，在示例配置中，10.x.x.x 网络表示公共网络。

要配置此示例场景，在创建 NetScaler CPX 容器时，在 Linux shell 提示符下运行以下命令：

```
docker run -dt -p 22 -p 80 -p 161/udp -p 8080:8080 --ulimit core=-1 --privileged=true cpx:6.2
```

通过使用 NetScaler Console 中的“作业”功能或使用 NITRO API 运行以下命令：

```
add service s1 10.102.29.100 HTTP 8081

add service s2 10.102.29.105 HTTP 8081

add service s3 10.102.29.105 HTTP 8082

add service s4 10.102.29.110 HTTP 8081

add service s5 10.102.29.110 HTTP 8082

add lb vserver cpx-vip HTTP 172.17.0.2 8080

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2

bind lb vserver cpx-vip s3

bind lb vserver cpx-vip s4
```

```
bind lb vserver cpx-vip s5
```

部署 NetScaler CPX，直接访问网络

在桥接网络模式下，您可以配置 NetScaler CPX 实例以直接访问网络。在这种情况下，传入流量将直接在 NetScaler CPX 虚拟服务器 IP (VIP) 上接收。

要启用此通信，您必须首先在 docker0 网桥上配置一个公共 IP 地址。然后，从网络端口 eth0 中删除该公共 IP 地址，并将网络端口绑定到 docker0 网桥。

通过添加两个服务来配置负载均衡，然后将网络公共 IP 地址配置为 NetScaler CPX 实例上的 VIP。客户端请求将直接在 VIP 上接收。

在示例配置中，10.x.x.x 网络表示公共网络。

要配置此场景，请在 Linux shell 提示符下运行以下命令：

```
ip addr add 10.102.29.100/24 dev docker0;

ip addr del 10.102.29.100/24 dev eth0;

brctl addif docker0 eth0;

ip route del default;

ip route add default via 10.102.29.1 dev docker0
```

通过使用 NetScaler Console 中的作业功能或使用 NITRO API，运行以下命令：

```
add service s1 172.17.0.8 http 80

add service s2 172.17.0.9 http 80

add lb vserver cpx-vip HTTP 10.102.29.102 80

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

使用 ConfigMap 在 Kubernetes 中配置 NetScaler CPX

在 Kubernetes 中，您可以使用 ConfigMap 配置 NetScaler CPX 实例。使用 ConfigMap，您可以在实例启动期间动态配置 NetScaler CPX 实例。

创建一个 `cpx.conf` 配置文件，其中包含 NetScaler 特定的配置以及您希望在 NetScaler CPX 实例上动态运行的 bash shell 命令。该配置文件结构需要两种类型的标签：`#NetScaler Commands` 和 `#Shell Commands`。在 `#NetScaler Commands` 标签下，您必须添加所有 NetScaler 命令，以在 NetScaler CPX 实例上配置 NetScaler 特定的配置。在 `#Shell Commands` 标签下，您必须添加希望在 NetScaler CPX 实例上运行的 shell 命令。

重要提示：

- 这些标签可以在配置文件中重复多次。
- 配置文件还可以包含注释。在注释前添加“#”字符。
- 这些标签不区分大小写。
- 如果在部署带有配置文件的 NetScaler CPX 容器时出现故障情况，故障将记录在 `ns.log` 文件中。
- NetScaler CPX 实例启动后，如果您更改 ConfigMap，更新的配置仅在 NetScaler CPX 实例重新启动时才应用。

以下是一个示例配置文件：

```
#NetScaler Commands

add lb vserver v1 http 1.1.1.1 80

add service s1 2.2.2.2 http 80

bind lb vserver v1 s1

#Shell Commands
```

```
touch /etc/a.txt

echo "this is a" > /etc/a.txt

#NetScaler Commands

add lb vserver v2 http

#Shell Commands

echo "this is a 1" >> /etc/a.txt

#NetScaler Commands

add lb vserver v3 http
```

创建配置文件后，您必须使用 `kubectl create configmap` 命令从配置文件创建 ConfigMap。

```
kubectl create configmap cpx-config --from-file=cpx.conf
```

在上面的示例中，您可以基于配置文件 `cpx.conf` 创建一个 ConfigMap，`cpx-config`。然后，您可以在用于部署 NetScaler CPX 实例的 YAML 文件中使用此 ConfigMap。

您可以使用 `kubectl get configmap` 命令查看创建的 ConfigMap。 `root@node1:~/yaml#`

```
kubectl get configmap cpx-config -o yaml
```

示例：

```
apiVersion: v1

data:

  cpx.conf: |

    #NetScaler Commands

    add lb vserver v1 http 1.1.1.1 80

    add service s1 2.2.2.2 http 80

    bind lb vserver v1 s1
```

```

#Shell Commands

touch /etc/a.txt

echo "this is a" > /etc/a.txt

echo "this is the file" >> /etc/a.txt

ls >> /etc/a.txt

#NetScaler Commands

add lb vserver v2 http

#Shell Commands

echo "this is a 1" >> /etc/a.txt

#NetScaler Commands

add lb vserver v3 http

#end of file

kind: ConfigMap

metadata:

  creationTimestamp: 2017-12-26T06:26:50Z

  name: cpx-config

  namespace: default

  resourceVersion: "8865149"

  selfLink: /api/v1/namespaces/default/configmaps/cpx-config

  uid: c1c7cb5b-ea05-11e7-914a-926745c10b02

```

您可以按如下方式在用于部署 NetScaler CPX 实例的 YAML 文件中指定创建的 ConfigMap `cpx-config` :

```
apiVersion: v1

kind: Pod

metadata:

  name: cpx-1

  labels:

    app: cpx-daemon

  annotations:

    NETSCALER_AS_APP: "True"

spec:

  hostNetwork: true

  containers:

  - name: cpx

    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.1-33.47"

    securityContext:

      privileged: true

    volumeMounts:

      - name: config-volume

        mountPath: /cpx/bootup_conf

    env:

      - name: "EULA"

        value: "yes"

      - name: "NS_NETMODE"
```

```
      value: "HOST"

    - name: "kubernetes_url"

      value: "https://10.90.248.101:6443"

    - name: "NS_MGMT_SERVER"

      value: "10.90.248.99"

    - name: "NS_MGMT_FINGER_PRINT"

      value:
"19:71:A3:36:85:0A:2B:62:24:65:0F:7E:72:CC:DC:AD:B8:BF:53:1E"

    - name: "NS_ROUTABLE"

      value: "FALSE"

    - name: "KUBERNETES_TASK_ID"

      valueFrom:

        fieldRef:

          fieldPath: metadata.name

      imagePullPolicy: Never

  volumes:

    - name: config-volume

      configMap:

        name: cpx-config
```

NetScaler CPX 实例部署并启动 ConfigMap 中指定的配置后，`cpx-config` 将应用于 NetScaler CPX 实例。

使用 ConfigMap 为 NetScaler CPX 配置自定义指标

您可以使用 ConfigMap 将自定义指标配置文件（JSON 格式）提供给 Kubernetes 中的 NetScaler CPX™ 实例。这允许您添加自定义指标架构文件，而无需覆盖包含 CPX 的默认指标配置文件（`schema.json` 和 `reference_schema.json`）。

在启动期间，NetScaler CPX 实例会将默认指标配置文件放置在 `/var/metrics_conf` 目录中。如果您在 `/cpx/custom_metrics_conf` 处挂载 ConfigMap，CPX 将验证每个 JSON 文件，并将有效文件复制到 `/var/metrics_conf` 中，与默认文件并存。

注意：

- 自定义 JSON 文件必须包含有效的 JSON。无效文件在启动期间将被跳过，并且错误将记录在 `boot.log` 中。
- 默认文件（`schema.json`、`reference_schema.json`）始终优先放置。
- 与默认文件同名的自定义文件将覆盖默认文件。

配置自定义指标

您可以通过创建 ConfigMap 并部署已挂载 ConfigMap 的 NetScaler CPX 实例来配置 NetScaler CPX 的自定义指标。

使用 CLI 进行配置

创建您要提供给 NetScaler CPX 实例的自定义指标配置文件（JSON 格式）。以下是一个示例自定义配置文件：

```
{  
  
  "counter": "http_requests_total",  
  
  "label": "method",  
  
  "description": "Total HTTP requests by method"
```

```
}
```

使用 `kubectl create configmap` 命令从文件创建 ConfigMap。例如，要基于配置文件 `custom_schema.json` 创建名为 `custom-metrics-conf` 的 ConfigMap：

```
kubectl create configmap custom-metrics-conf --from-  
file=custom_schema.json=./custom_schema.json
```

使用 `kubectl get configmap` 命令查看创建的 ConfigMap。

```
kubectl get configmap custom-metrics-conf -o yaml
```

以下示例显示了示例输出：

```
apiVersion: v1  
  
data:  
  
  custom_schema.json: |  
  
    {  
  
      "counter": "http_requests_total",  
  
      "label": "method",  
  
      "description": "Total HTTP requests by method"  
  
    }  
  
kind: ConfigMap  
  
metadata:  
  
  creationTimestamp: 2026-05-14T06:26:50Z  
  
  name: custom-metrics-conf  
  
  namespace: default
```

```
resourceVersion: "8865149"
```

```
uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
```

使用 YAML 文件进行配置

在用于部署 NetScaler CPX 实例的 YAML 文件中指定创建的 ConfigMap `custom-metrics-conf`。以下示例配置展示了如何在部署 YAML 中包含 ConfigMap：

```
apiVersion: v1

kind: Pod

metadata:

  name: cpx-1

  labels:

    app: cpx-daemon

  annotations:

    NETSCALER_AS_APP: "True"

spec:

  containers:

    - name: cpx

      image: "quay.io/netScaler/netScaler-cpx:14.1-72.56"

      securityContext:

        privileged: true

  volumeMounts:

    - name: config-volume

      mountPath: /cpx/bootup_conf
```

```
- name: custom-metrics-volume

  mountPath: /cpx/custom_metrics_conf

env:

- name: "EULA"

  value: "yes"

  imagePullPolicy: IfNotPresent

volumes:

- name: config-volume

  configMap:

    name: cpx-config

- name: custom-metrics-volume

  configMap:

    name: custom-metrics-conf
```

部署 NetScaler CPX 实例。实例部署并启动后，默认指标配置文件 (`schema.json` 和 `reference_schema.json`) 将放置在 `/var/metrics_conf` 中。从挂载在 `/cpx/custom_metrics_conf` 的 ConfigMap 中的自定义 JSON 文件将被验证并复制到 `/var/metrics_conf` 中。

部署 NetScaler CPX 作为 Kubernetes 节点的本地 DNS 缓存

Kubernetes 集群中的应用程序 Pod 依赖 DNS 与其他应用程序 Pod 进行通信。来自 Kubernetes 集群内部应用程序的 DNS 请求由 Kubernetes DNS (kube-dns) 处理。由于微服务架构的广泛采用，Kubernetes 集群内部的 DNS 请求速率正在增加。因此，Kubernetes DNS (kube-dns) 负担过重。现在，您可以在每个 Kubernetes 节点上部署 NetScaler CPX 作为本地 DNS 缓存，并将节点中应用程序 Pod 的 DNS 请求转发到 NetScaler CPX。因此，您可以更快地解析 DNS 请求，并显著减轻 Kubernetes DNS 的负载。

为了部署 NetScaler CPX，可以使用 Kubernetes DaemonSet 实体在 Kubernetes 集群中的每个节点上调度 NetScaler CPX Pod。Kubernetes DaemonSet 确保集群中每个 Kubernetes 节点上都有一个 NetScaler CPX 实例。为了使应用程序 Pod 将流量直接发送到 CPX DNS Pod，您需要创建一个 Kubernetes 服务，并将 NetScaler CPX Pod 作为其端点。此服务的集群 IP 用作应用程序 Pod 的 DNS 端点。为确保应用程序 Pod 使用 NetScaler CPX 服务集群 IP 地址进行 DNS 解析，您需要在每个节点上使用 NetScaler CPX 服务集群 IP 更新 kubelet 配置文件。

引入以下环境变量以支持将 NetScaler CPX 部署为 NodeLocal DNS 缓存：

- **KUBE_DNS_SVC_IP**：指定 **kube-dns** 服务的集群 IP 地址，这是在 NetScaler CPX Pod 上触发配置的强制参数。当 NetScaler CPX 缓存中没有 DNS 查询响应时，NetScaler CPX Pod 会将 DNS 查询定向到此 IP 地址。
- **CPX_DNS_SVC_IP**：指定 NetScaler CPX 服务的集群 IP 地址。**CPX_DNS_SVC_IP** 环境变量用于在节点上配置本地 DNS。当您配置此变量时，会添加一个 **iptables** 规则，将源自应用程序 Pod 的 DNS 请求定向到节点内的本地 NetScaler CPX Pod。
- **NS_DNS_FORCE_TCP**：此环境变量强制对 DNS 请求使用 TCP，即使查询是通过 UDP 接收的。
- **NS_DNS_EXT_RESOLV_IP**：指定外部名称服务器的 IP 地址，以将特定域的 DNS 请求定向到该服务器。
- **NS_DNS_MATCH_DOMAIN**：指定要匹配的外部域字符串，以将查询定向到外部名称服务器。

在节点上部署 NetScaler CPX 作为 DNS 缓存

将 NetScaler CPX 部署为 Kubernetes 集群的本地 DNS 缓存包括以下任务：

在主节点上：

- 创建 Kubernetes 服务，并将 NetScaler CPX Pod 作为其端点
- 创建 ConfigMap，用于为 NetScaler CPX Pod 定义环境变量
- 使用 Kubernetes DaemonSet 在 Kubernetes 集群中的每个节点上调度 NetScaler CPX Pod。

在工作节点上：

- 修改 kubelet 配置文件，使用 NetScaler CPX 服务的集群 IP 地址将 DNS 请求转发到 NetScaler CPX。

Kubernetes 主节点上的配置

在 Kubernetes 主节点上执行以下步骤，以将 NetScaler CPX 部署为节点的本地 DNS 缓存：

1. 使用 `cpx_dns_svc.yaml` 文件创建一个以 NetScaler CPX Pod 作为端点的服务。

```
kubectl apply -f cpx_dns_svc.yaml
```

`cpx_dns_svc.yaml` 文件提供如下：

```
apiVersion: v1

kind: Service

metadata:

  name: cpx-dns-svc

  labels:

    app: cpxd

spec:

  ports:

    - protocol: UDP

      port: 53

      name: dns

    - protocol: TCP
```

```
port: 53

name: dns-tcp

selector:

app: cpx-daemon
```

2. 获取 NetScaler CPX 服务的 IP 地址。

```
kubectl get svc cpx-dns-svc
```

3. 获取 Kube DNS 服务的 IP 地址。

```
kubectl get svc -n kube-system
```

4. 创建一个 ConfigMap，用于定义 NetScaler CPX Pod 的环境变量。这些环境变量用于传递 NetScaler CPX 服务和 Kube DNS 服务的 IP 地址。在此步骤中，使用文件中指定为数据（键值对）的环境变量创建了一个示例 ConfigMap `cpx-dns-cache`。

```
kubectl create configmap cpx-dns-cache --from-file <path-to-file>
```

以下是一个示例文件，其中环境变量为键值对。

```
CPX_DNS_SVC_IP: 10.111.95.145

EULA: "yes"

KUBE_DNS_SVC_IP: 10.96.0.10

NS_CPX_LITE: "1"

NS_DNS_EXT_RESLV_IP: 10.102.217.142

NS_DNS_MATCH_DOMAIN: citrix.com

PLATFORM: CP1000
```

以下是一个示例 ConfigMap：

```
apiVersion: v1

data:

  CPX_DNS_SVC_IP: 10.111.95.145

  EULA: "yes"

  KUBE_DNS_SVC_IP: 10.96.0.10

  NS_CPX_LITE: "1"

  NS_DNS_EXT_RESLV_IP: 10.102.217.142

  NS_DNS_MATCH_DOMAIN: citrix.com

  PLATFORM: CP1000

kind: ConfigMap

metadata:

  creationTimestamp: "2019-10-15T07:45:54Z"

  name: cpx-dns-cache

  namespace: default

  resourceVersion: "8026537"

  selfLink: /api/v1/namespaces/default/configmaps/cpx-dns-cache

  uid: 8d06f6ee-133b-4e1a-913c-9963cbf4f48
```

5. 在主节点上为 NetScaler CPX 创建一个 Kubernetes DaemonSet。

```
kubectl apply -f cpx_daemonset.yaml
```

`cpx_daemonset.yaml` 文件提供如下：

```
apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: cpx-daemon

  labels:

    app: cpxd

spec:

  selector:

    matchLabels:

      app: cpx-daemon

  template:

    metadata:

      labels:

        app: cpx-daemon

    spec:

      containers:

        - name: cpxd

          imagePullPolicy: IfNotPresent

          image: localhost:5000/dev/cpx

          volumeMounts:

            - mountPath: /netns/default/

      name: test-vol
```

```

    ports:

      - containerPort: 53

    envFrom:

      - configMapRef:

          name: cpx-dns-cache

    securityContext:

      privileged: true

      allowPrivilegeEscalation: true

    capabilities:

      add: ["NET_ADMIN"]

    volumes:

      - name: test-vol

        hostPath:

          path: /proc/1/ns

          type: Directory

```

Kubernetes 集群中工作节点上的配置

在主节点上完成配置后，在工作节点上执行以下步骤：

1. 修改 kubelet 配置文件，以便应用程序 Pod 可以使用 NetScaler CPX 服务集群 IP 进行 DNS 解析，方法如下：
 - 按照 [重新配置节点的 kubelet](#) 中的步骤，并按以下格式修改 `--cluster-dns` 参数值。

```
--cluster-dns=&lt;CPX_DNS_SVC_IP>,&lt;KUBE_DNS_SVC_IP>
```

或者

- 编辑 `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` 文件，并使用以下步骤修改 `--cluster-dns` 参数。

- a. 编辑 kubelet 配置，并为 `--cluster-dns` 参数指定 NetScaler CPX 服务的集群 IP 地址和 `kube-dns` 服务 IP 地址。

```
root@node:~# cat /etc/systemd/system/kubelet.service.d/10-  
kubeadm.conf | grep KUBELET_DNS_ARGS  
  
Environment="KUBELET_DNS_ARGS=--cluster-  
dns=10.111.95.145,10.96.0.10 --cluster-domain=cluster.local"  
  
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS  
$KUBELET_CONFIG_ARGS $KUBELET_DNS_ARGS
```

- b. 使用以下命令重新加载节点的 kubelet：

```
# systemctl daemon-reload  
  
# service kubelet restart
```

在 Google Compute Engine 上部署 NetScaler CPX 代理

本部署指南介绍了如何在 Google Cloud 的 Google Compute Engine (GCE) 上使用 Docker 部署 NetScaler CPX，并在企业网络中运行 NetScaler Console。在此部署中，安装在 GCE 上的 NetScaler CPX 对两个后端服务器进行负载均衡，NetScaler Console 提供许可和分析解决方案。

NetScaler CPX 是一种基于容器的代理，支持完整的第 7 层功能、SSL 卸载、多种协议和 NITRO API。NetScaler Console 提供管理、许可和分析解决方案。作为许可服务器，NetScaler Console 为在本地或云中运行的 NetScaler CPX 实例提供授权。

CPX 和 CPX Express 是相同的映像。当您使用 NetScaler Console 许可并安装 CPX 映像时，Docker App Store 中的 CPX 映像（版本 11 或 12）将成为一个完整的 CPX 实例。如果没有许可证，CPX 映像将成为一个 CPX Express 实例，支持 20 Mbps 和 250 个 SSL 连接。

先决条件

- 2 GB 内存和 1 个 vCPU 专用于 NetScaler CPX
- 可从 GCE 获取的 Docker 开源
- 在本地运行的 NetScaler Console，通过互联网或 VPN 连接到 GCE

注意

有关如何部署 NetScaler Console 的信息，请参阅[部署 NetScaler Console](#)。

配置步骤

您必须执行以下步骤才能配置此部署。

1. 在 GCE VM 上安装 Docker。
2. 配置与 Docker 实例的远程 API 通信。
3. 安装 NetScaler CPX 映像。
4. 创建一个 CPX 实例。
5. 通过 NetScaler Console 许可 NetScaler CPX。

6. 在 NetScaler CPX 上配置负载均衡服务并验证配置。
 - a. 安装 NGINX Web 服务器。
 - b. 配置 NetScaler CPX 进行负载均衡，并验证负载在两个 Web 服务之间的分配。

步骤 1：在 GCE VM 上安装 Docker

在 GCE 中，创建一个 Linux Ubuntu VM。然后，使用以下示例中显示的命令在 VM 上安装 Docker：

```
$ sudo curl -ssl https://get.docker.com/ | sh

% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed

0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve
host: xn--ssl-1n0a

100 17409 100 17409 0 0 21510 0 --:--:-- --:--:-- --:--:-- 21492

apparmor is enabled in the kernel and apparmor utils were already
installed

\+ sudo -E sh -c apt-key add -

\+ echo -----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v1

mQINBFWln24BEADrBl5p99uKh8+rpvqJ48u4eTtjeXAWbslJotmC/CakbNSqOb9o
ddfzRvGVeJVERt/Q/m1vEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
mYdoe/BJ/J/6rH0jq70mk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyU/jPpWFK28TR8yfVlzYFwibj5WK
dHM7ZTq1A1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccy0i+ULU0z8mHUJ3iEMIrPT
X+80KaN/ZjibfsBOCjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
```

H0oyC6qVe0gKrRiKxzymj0FIMLru/iFF5pSwcQB7PYlt8J0G80lAcPr6VCiN+4c
NKv03SdvA69dC0j79Pu09IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHhM1fDeCqkZ
hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Za1tn0svoGs+wX3Z/H6/8P5anog43U
65c0A+64Jj00rNdr8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+c0MHUU4HA7iM
zHrouAdYeTZeZEQ0A7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFncTKdv+rwARAQAB
tDdEb2NrZXIqUmVsZWZzZSBub29sIchyZWxlYXNlZG9ja2VyKSA8ZG9ja2VyQGRv
Y2t1ci5jb20+iQIcBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4Syo0CV
cnybl18vzwBEcp4VCRbXvHv0Xty1gccVIV8/aJqNKgBV97lY3vrp0yiIeB8ETQeg
srxFE7t/Gz0rsL0bqfLEHdmn5iBJRkhLfCpzje0nyB3Z0IJB6Uog0/msQVYe5CXJ
16uwr0AmoiCBLrVlDAktxVh9RWch0l0KZR2FpHu8h+uM0/zySqIidlYfLa3y5oH
scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdcAHR+bXhA/RW8QHeeMFliWw
7Z2jYHyuHmDnWG2yUrnCqAJTrwV+OfKRIZZJFBs4e88ru5h2ZIXdRepw/+C0Yj34
LyzxR2cxr2u/xvxwXCkSMe7F4KZAphD+1ws61FhnUMi/PERMYfTFuvPrCkq4gyBj
t3fFpZ2NR/fKW87Q0eVcn1ivXl9id3MMs9KXJsg7Qast7mCsee2VIFsxrkFQ2jNp
D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaH0
w1zblWy/CFr6VF1963jrjJgag0G6tNtBZLrc1ZgWh0QpeZZ5Lbvz2ZA5CqRrFAVc
wPNW1f0bFIRtqV6vuVluF0PCMAAn0nqR02w9t17ivQj03oVN0mbQi9vjuExXh1Yo
ScVeti06LSm1QfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBlAAoJEFyzYeVS
+w0QeH0QAI6btAfYwYPuAjrUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+1/R08nu0d
sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
NydGFzmeOKbAlrxIMECnSTG/TkFVY09Nt1v9vSN2BupmTagTRErxLZKnVSWRzp+X

```
\-----END PGP PUBLIC KEY BLOCK-----
```

```
OK
```

```
\+ sudo -E sh -c mkdir -p /etc/apt/sources.list.d
```

```
\+ dpkg --print-architecture
```

```
\+ sudo -E sh -c echo deb \\\[arch=amd64\\\] https://  
apt.dockerproject.org/repo ubuntu-yakkety main \> /etc/apt/sources.list.d/  
docker.list
```

```
\+ sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-  
engine
```

```
Hit:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety InRelease
```

```
Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates  
InRelease \[102 kB\]
```

```
Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports  
InRelease \[102 kB\]
```

```
Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted  
Sources \[5,376 B\]
```

```
Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse  
Sources \[181 kB\]
```

```
Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe  
Sources \[8,044 kB\]
```

```
Get:7 http://archive.canonical.com/ubuntu yakkety InRelease \[11.5 kB\]
```

```
Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease \[102  
kB\]
```

```
Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease \[47.3  
kB\]
```

```
Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main Sources
```

\[903 kB\]

Get:11 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
restricted Sources \[2,688 B\]

Get:12 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
universe Sources \[57.9 kB\]

Get:13 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
multiverse Sources \[3,172 B\]

Get:14 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/main
Sources \[107 kB\]

Get:15 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/main
amd64 Packages \[268 kB\]

Get:16 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/main
Translation-en \[122 kB\]

Get:17 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
universe amd64 Packages \[164 kB\]

Get:18 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
universe Translation-en \[92.4 kB\]

Get:19 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
multiverse amd64 Packages \[4,840 B\]

Get:20 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-updates/
multiverse Translation-en \[2,708 B\]

Get:21 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-backports/
universe Sources \[2,468 B\]

Get:22 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-backports/
main Sources \[2,480 B\]

Get:23 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-backports/
main amd64 Packages \[3,500 B\]

Get:24 <http://us-west1.gce.archive.ubuntu.com/ubuntu> yakkety-backports/
universe amd64 Packages \[3,820 B\]

```
Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe Translation-en \[1,592 B\  
  
Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64 Packages  
\[2,480 B\  
  
Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources  
\[47.7 kB\  
  
Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64  
Packages \[2,453 B\  
  
Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe Sources  
\[20.7 kB\  
  
Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse  
Sources \[1,140 B\  
  
Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted  
Sources \[2,292 B\  
  
Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64  
Packages \[150 kB\  
  
Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main  
Translation-en \[68.0 kB\  
  
Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe amd64  
Packages \[77.2 kB\  
  
Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe  
Translation-en \[47.3 kB\  
  
Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse amd64  
Packages \[2,832 B\  
  
Fetched 10.8 MB in 2s (4,206 kB/s)  
  
Reading package lists... Done  
  
Reading package lists...  
  
Building dependency tree...
```

Reading state information...

The following additional packages will be installed:

aufs-tools cgroupfs-mount libltdl7

The following NEW packages will be installed:

aufs-tools cgroupfs-mount docker-engine libltdl7

0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.

Need to get 21.2 MB of archives.

After this operation, 111 MB of additional disk space will be used.

Get:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe amd64 aufs-tools amd64 1:3.2+20130722-1.1ubuntu1 \[92.9 kB\]

Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe amd64 cgroupfs-mount all 1.3 \[5,778 B\]

Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main amd64 libltdl7 amd64 2.4.6-1 \[38.6 kB\]

Get:4 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64 docker-engine amd64 17.05.0~ce-0~ubuntu-yakkety \[21.1 MB\]

Fetchd 21.2 MB in 1s (19.8 MB/s)

Selecting previously unselected package aufs-tools.

(Reading database ... 63593 files and directories currently installed.)

Preparing to unpack .../aufs-tools_1%3a3.2+20130722-1.1ubuntu1_amd64.deb ...

Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...

Selecting previously unselected package cgroupfs-mount.

Preparing to unpack .../cgroupfs-mount_1.3_all.deb ...

```
Unpacking cgroupfs-mount (1.3) ...

Selecting previously unselected package libltdl7:amd64.

Preparing to unpack .../libltdl7\2.4.6-1\_amd64.deb ...

Unpacking libltdl7:amd64 (2.4.6-1) ...

Selecting previously unselected package docker-engine.

Preparing to unpack .../docker-engine\_17.05.0~ce-0~ubuntu-
yakkety\_amd64.deb ...

Unpacking docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...

Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...

Processing triggers for ureadahead (0.100.0-19) ...

Setting up cgroupfs-mount (1.3) ...

Processing triggers for libc-bin (2.24-3ubuntu2) ...

Processing triggers for systemd (231-9ubuntu4) ...

Setting up libltdl7:amd64 (2.4.6-1) ...

Processing triggers for man-db (2.7.5-1) ...

Setting up docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...

Created symlink /etc/systemd/system/multi-user.target.wants/docker.service
→ /lib/systemd/system/docker.service.

Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /
lib/systemd/system/docker.socket.

Processing triggers for ureadahead (0.100.0-19) ...

Processing triggers for libc-bin (2.24-3ubuntu2) ...

Processing triggers for systemd (231-9ubuntu4) ...
```

```
\+ sudo -E sh -c docker version
```

Client:

Version: 17.05.0-ce

API version: 1.29

Go version: go1.7.5

Git commit: 89658be

Built: Thu May 4 22:15:36 2017

OS/Arch: linux/amd64

Server:

Version: 17.05.0-ce

API version: 1.29 (minimum version 1.12)

Go version: go1.7.5

Git commit: 89658be

Built: Thu May 4 22:15:36 2017

OS/Arch: linux/amd64

Experimental: false

If you would like to use Docker as a non-root user, you should now consider

adding your user to the "docker" group with something like:

```
sudo usermod -aG docker albert\_lee
```

Remember that you will have to log out and back in for this to take effect.

WARNING: Adding a user to the "docker" group will grant the ability to run containers which can be used to obtain root privileges on the docker host.

Refer to <https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface>

for more information.

\$

```
**$ sudo docker info**
```

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 17.05.0-ce

Storage Driver: aufs

Root Dir: /var/lib/docker/aufs

Backing Filesystem: extfs

Dirs: 0

Dirperm1 Supported: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9048e5e50717ea4497b757314bad98ea3763c145

runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228

init version: 949e6fa

Security Options:

apparmor

seccomp

Profile: default

Kernel Version: 4.8.0-51-generic

Operating System: Ubuntu 16.10

OSType: linux

Architecture: x86_64

CPUs: 1

Total Memory: 3.613GiB

Name: docker-7

ID: R5TW:VKXK:EKGR:GHWM:UNU4:LPJH:IQY5:X77G:NNRQ:HWBY:LIUD:4ELQ

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Experimental: false

Insecure Registries:

127.0.0.0/8

Live Restore Enabled: false

WARNING: No swap limit support

\$

****\$ sudo docker images****

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

\$

****\$ sudo docker ps****

CONTAINER	ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
\$							

步骤 2：配置与 Docker 实例的远程 API 通信

打开端口 4243 以便与 Docker 实例进行 API 通信。NetScaler Console 需要此端口才能与 Docker 实例通信。

```

**cd /etc/systemd/system**

**sudo vi docker-tcp.socket**

**cat docker-tcp.socket**

[Unit\

**Description=Docker Socket for the API

\[Socket\

ListenStream=4243

BindIPv6only=both

Service=docker.service

\[Install\

WantedBy=sockets.target**

$ **sudo systemctl enable docker-tcp.socket**

Created symlink /etc/systemd/system/sockets.target.wants/docker-tcp.socket
→ /etc/systemd/system/docker-tcp.socket.

```

```

**sudo systemctl enable docker.socket**

**sudo systemctl stop docker**

**sudo systemctl start docker-tcp.socket**

**sudo systemctl start docker**

$ **sudo systemctl status docker**

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
preset: enabled)

Active: **active (running)** since Wed 2017-05-31 12:52:17 UTC; 2s ago

Docs: https://docs.docker.com

Main PID: 4133 (dockerd)

Tasks: 16 (limit: 4915)

Memory: 30.1M

CPU: 184ms

CGroup: /system.slice/docker.service

└─4133 /usr/bin/dockerd -H fd://

└─4137 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-
containerd.sock --metrics-interval=0 --start-timeout 2m -

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.300890402Z" level=warning msg="Your kernel does
not support cgroup rt peri

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.301079754Z" level=warning msg="Your kernel does
not support cgroup rt runt

```

```
May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.301681794Z" level=info msg="Loading containers:
start."

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.417539064Z" level=info msg="Default bridge
(docker0) is assigned with an I

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.465011600Z" level=info msg="Loading containers:
done."

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.484747909Z" level=info msg="Daemon has completed
initialization"

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.485119478Z" level=info msg="Docker daemon"
commit=89658be graphdriver=aufs

May 31 12:52:17 docker-7 systemd\[1\]: Started Docker Application
Container Engine.

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.503832254Z" level=info msg="API listen on /var/
run/docker.sock"

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.504061522Z" level=info msg="API listen on \[::
\]:4243"

$

(external)$ **curl 104.199.209.157:4243/version**

{"Version":"17.05.0-
ce","ApiVersion":"1.29","MinAPIVersion":"1.12","GitCommit":"89658be","GoVe
rsion":"go1.7.5","Os":"linux","Arch":"amd64","KernelVersion":"4.8.0-52-
generic","BuildTime":"2017-05-04T22:15:36.071254972+00:00"}

(external)$
```

步骤 3：安装 NetScaler CPX 映像

从 Docker App Store 获取 NetScaler CPX 映像。CPX Express 和 CPX 具有相同的映像。但是，当您使用 NetScaler Console 许可并安装 CPX 映像时，该映像将成为一个具有 1 Gbps 性能的完整 CPX 实例。如果没有许可证，该映像将成为一个 CPX Express 实例，支持 20 Mbps 和 250 个 SSL 连接。

```
$ **sudo docker pull store/citrix/citrixadccpx:13.0-36.29**
```

```
13.0-36.29: Pulling from store/citrix/citrixadccpx
```

```
4e1f679e8ab4: Pull complete
```

```
a3ed95caeb02: Pull complete
```

```
2931a926d44b: Pull complete
```

```
362cd40c5745: Pull complete
```

```
d10118725a7a: Pull complete
```

```
1e570419a7e5: Pull complete
```

```
d19e06114233: Pull complete
```

```
d3230f008ffd: Pull complete
```

```
22bdb10a70ec: Pull complete
```

```
1a5183d7324d: Pull complete
```

```
241868d4ebff: Pull complete
```

```
3f963e7ae2fc: Pull complete
```

```
fd254cf1ea7c: Pull complete
```

```
33689c749176: Pull complete
```

```
59c27bad28f5: Pull complete
```

```
588f5003e10f: Pull complete

Digest:
sha256:31a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe

Status: Downloaded newer image for store/citrix/citrixadccpx:13.0-36.29

$

$ **sudo docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

store/citrix/citrixadccpx:13.0-36.29 6fa57c38803f 3 weeks ago 415MB

$
```

步骤 4：创建 NetScaler CPX 实例

在 Docker 主机上安装 NetScaler CPX 映像。打开特定服务的端口，如以下示例所示，并为 NetScaler Console 指定一个 IP 地址：

```
bash-2.05b\# **CHOST=${1:-localhost}**

bash-2.05b\# **echo | openssl s_client -connect $CHOST:443 | openssl x509
-fingerprint -noout | cut -d'=' -f2**

depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU =
Internal, CN = Test Only Cert

verify error:num=18:self signed certificate

verify return:1

depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU =
Internal, CN = Test Only Cert

verify return:1

DONE
```

```
24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4
```

```
bash-2.05b\#
```

```
$ **sudo docker run -dt -p 50000:88 -p 5080:80 -p 5022:22 -p 5443:443 -p
5163:161/udp -e NS\_HTTP\_PORT=5080 -e NS\_HTTPS\_PORT=5443 -e
NS\_SSH\_PORT=5022 -e NS\_SNMP\_PORT=5163 -e EULA=yes -e
LS\_IP=xx.xx.xx.xx -e PLATFORM=CP1000 --privileged=true --ulimit core=-1
-e NS\_MGMT\_SERVER=xx.xx.xx.xx:xxxx -e
NS\_MGMT\_FINGER\_PRINT=24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC
:1E:86:A4 --env NS\_ROUTABLE=false --env HOST=104.199.209.157 store/
citrix/citrixadccpx:13.0-36.29**
```

```
44ca1c6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
```

```
$
```

```
$ **sudo docker ps**
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

```
44ca1c6c0907 store/citrix/citrixadccpx:13.0-36.29 "/bin/sh -c 'bash ..."
19 seconds ago Up 17 seconds 0.0.0.0:5022->22/tcp, 0.0.0.0:5080->80/tcp,
0.0.0.0:50000->88/tcp, 0.0.0.0:5163->161/udp, 0.0.0.0:5443->443/tcp
gifted\_perlman
```

```
$
```

```
$ **ssh -p 5022 root@localhost**
```

```
root@localhost's password:
```

```
Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)
```

```
\* Documentation: https://www.citrix.com/
```

```
Last login: Mon Jun 5 18:58:51 2017 from xx.xx.xx.xx

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\# **cli\_script.sh 'show ns ip'**

exec: show ns ip

IpAddress Traffic Domain Type Mode Arp Icmp Vserver State

\-----
1\) 172.17.0.2 0 NetScaler IP Active Enabled Enabled NA Enabled

2\) 192.0.0.1 0 SNIP Active Enabled Enabled NA Enabled

Done

root@44ca1c6c0907:~\# **cli\_script.sh 'show licenseserver'**

exec: show licenseserver

1\) ServerName: xx.xx.xx.xxPort: 27000 Status: 1 Grace: 0 Gptimeleft: 0

Done

root@44ca1c6c0907:~\# cli\_script.sh 'show capacity'

exec: show capacity

Actualbandwidth: 1000 Platform: CP1000 Unit: Mbps Maxbandwidth: 3000
Minbandwidth: 20 Instancecount: 0

Done

root@44ca1c6c0907:~\#

$ **sudo iptables -t nat -L -n**
```

```
Chain PREROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
```

```
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
```

```
Chain POSTROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
```

```
MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
```

```
Chain DOCKER (2 references)
```

```
target prot opt source destination
```

```
RETURN all -- 0.0.0.0/0 0.0.0.0/0

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443

DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88

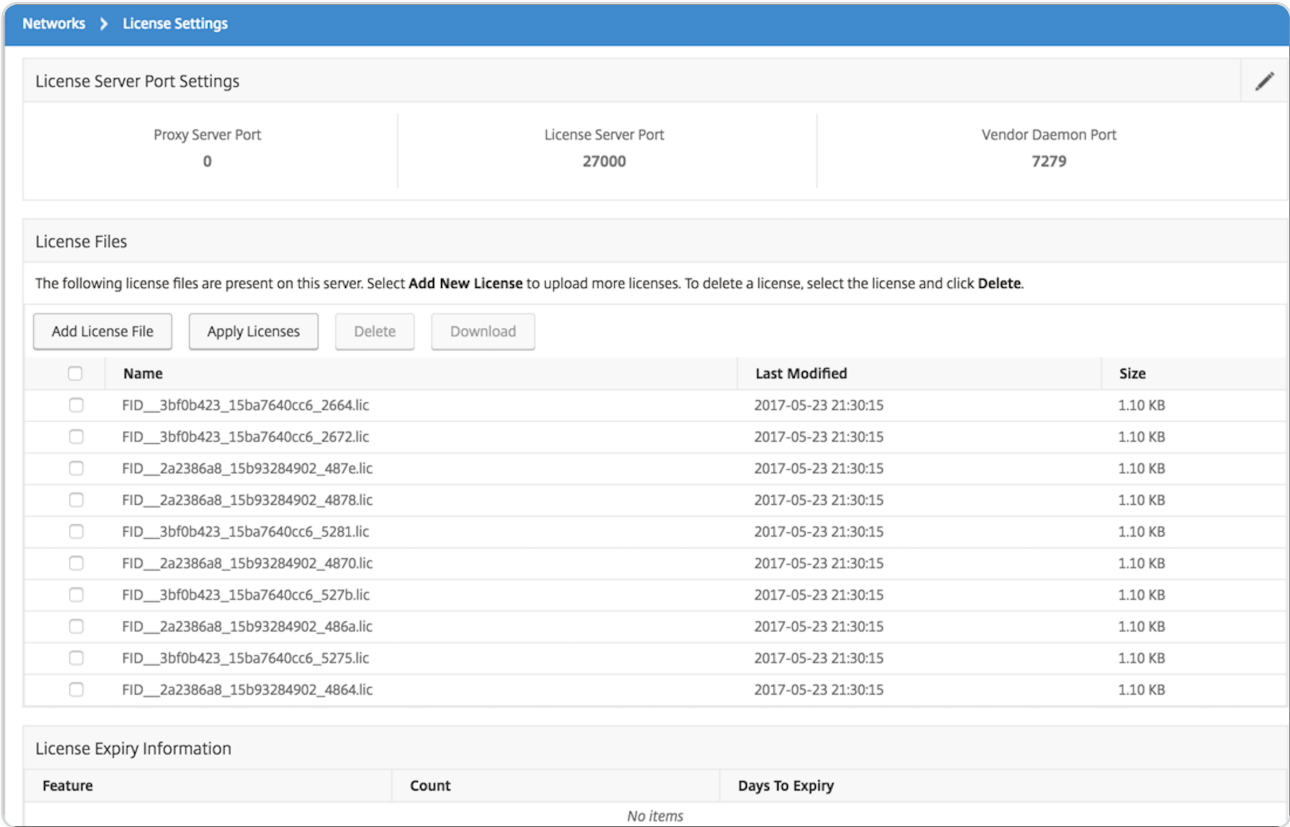
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22

$
```

步骤 5：通过 NetScaler Console 许可 NetScaler CPX

假设 NetScaler Console 在本地运行，您应该能够验证 NetScaler CPX 是否正在与 NetScaler Console 通信并发送信息。以下图像显示 NetScaler CPX 从 NetScaler Console 检索许可证。



Networks > Instances > NetScaler CPX

NetScaler CPX

Instances 3 Docker Host 0

Add Provision Delete Dashboard Reboot Action Search ⌵ ⚙

	IP Address	Host Name	State	Docker Host	Port Range	SSH Port	HTTP Port	HTTPS Port	SNMP Port
☐	172.17.0.2	-NA-	● Out of Service	104.196.190.229		32770	32769	32768	32768
☐	172.17.0.5	-NA-	● Out of Service	10.10.15.159	88-88	32785	32784	32783	32773
☐	172.17.0.2	-NA-	● Up	104.199.209.157		5022	5080	5443	5163

Networks > License Settings > CPX Licenses

CPX Licenses

Instances

10.0%

Total 10 Used 1

The following instances are consuming Instance license.

Name	IP Address	Instance Type	Allocation Status	Allocated Capacity
e516b1b61939	172.17.0.2	NetScaler CPX	Not available ⓘ	1

步骤 6: 在 NetScaler CPX 上配置负载均衡服务并验证配置

首先，在 Docker 主机上安装 NGINX Web 服务器。然后，在 NetScaler CPX 上配置负载均衡以平衡两个 Web 服务器的负载，并测试配置。

安装 NGINX Web 服务器

使用以下示例中显示的命令安装 NGINX Web 服务器。

```
$ sudo docker pull nginx

Using default tag: latest

latest: Pulling from library/nginx

Digest:
sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268

Status: Image is up to date for nginx:latest
```

```
**$ sudo docker run -d -p 81:80 nginx**
```

```
098a77974818f451c052ecd172080a7d45e446239479d9213cd4ea6a3678616f
```

```
**$ sudo docker run -d -p 82:80 nginx**
```

```
bbdac2920bb4085f70b588292697813e5975389dd546c0512daf45079798db65
```

```
**$ sudo iptables -t nat -L -n**
```

```
Chain PREROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
```

```
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
```

```
Chain POSTROUTING (policy ACCEPT)
```

```
target prot opt source destination

MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443

MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22

MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80

MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80
```

Chain DOCKER (2 references)

```
target prot opt source destination

RETURN all -- 0.0.0.0/0 0.0.0.0/0

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443

DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80

$
```

配置 NetScaler CPX 以进行负载均衡并验证负载到两个 Web 服务的分配

```
$ **ssh -p 5022 root@localhost**

root@localhost's password:

Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)

\* Documentation: https://www.citrix.com/

Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\# **cli\_script.sh "add service web1 172.17.0.3 HTTP
80"**

exec: add service web1 172.17.0.3 HTTP 80

Done

root@44ca1c6c0907:~\# **cli\_script.sh "add service web2 172.17.0.4 HTTP
80"**

exec: add service web2 172.17.0.4 HTTP 80

Done

root@44ca1c6c0907:~\# **cli\_script.sh "add lb vserver cpx-vip HTTP
172.17.0.2 88"**

exec: add lb vserver cpx-vip HTTP 172.17.0.2 88

Done
```

```
root@44ca1c6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web1"**
```

```
exec: bind lb vserver cpx-vip web1
```

```
Done
```

```
root@44ca1c6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web2"**
```

```
exec: bind lb vserver cpx-vip web2
```

```
Done
```

```
root@44ca1c6c0907:~\#
```

```
root@44ca1c6c0907:~\# **cli\_script.sh 'show lb vserver cpx-vip'**
```

```
exec: show lb vserver cpx-vip
```

```
cpx-vip (172.17.0.2:88) - HTTP Type: ADDRESS
```

```
State: UP
```

```
Last state change was at Mon Jun 5 19:01:49 2017
```

```
Time since last state change: 0 days, 00:00:42.620
```

```
Effective State: UP
```

```
Client Idle Timeout: 180 sec
```

```
Down state flush: ENABLED
```

```
Disable Primary Vserver On Down : DISABLED
```

```
Appflow logging: ENABLED
```

```
Port Rewrite : DISABLED
```

```
No. of Bound Services : 2 (Total) 2 (Active)
```

Configured Method: LEASTCONNECTION

Current Method: Round Robin, Reason: A new service is bound BackupMethod: ROUNDROBIN

Mode: IP

Persistence: NONE

Vserver IP and Port insertion: OFF

Push: DISABLED Push VServer:

Push Multi Clients: NO

Push Label Rule: none

L2Conn: OFF

Skip Persistency: None

Listen Policy: NONE

IcmpResponse: PASSIVE

RHISTate: PASSIVE

New Service Startup Request Rate: 0 PER_SECOND, Increment Interval: 0

Mac mode Retain Vlan: DISABLED

DBS_LB: DISABLED

Process Local: DISABLED

Traffic Domain: 0

TROFS Persistence honored: ENABLED

Retain Connections on Cluster: NO

```
2\) web1 (172.17.0.3: 80) - HTTP State: UP Weight: 1
```

```
3\) web2 (172.17.0.4: 80) - HTTP State: UP Weight: 1
```

```
Done
```

```
root@44ca1c6c0907:~\#
```

```
(external)$ **curl 104.199.209.157:50000**
```

```
\<!DOCTYPE html\>
```

```
\<html\>
```

```
\<head\>
```

```
\<title\>Welcome to nginx\!\</title\>
```

```
\<style\>
```

```
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

\nginx.org\</a\>.\<br/\>

Commercial support is available at

\nginx.com\</a\>.\</p\>

\<p\>\<em\>Thank you for using nginx.\</em\>\</p\>

\</body\>

\</html\>

(external)\$

```
(external)$ for i in {1..100} ; **do curl http://104.199.209.157:50000
-o /dev/null ; done**
```

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

					Dload	Upload	Total	Spent	Left
Speed									

100	612	100	612	0	0	1767	0	--:--:--	--:--:--	--:--:--
1768										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1893	0	--:--:--	--:--:--	--:--:--
1894										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1884	0	--:--:--	--:--:--	--:--:--
1883										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1917	0	--:--:--	--:--:--	--:--:--

1924

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1877	0	--:--:--	--:--:--	--:--:--
1883										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1852	0	--:--:--	--:--:--	--:--:--
1848										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1860	0	--:--:--	--:--:--	--:--:--
1865										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1887	0	--:--:--	--:--:--	--:--:--
1888										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1802	0	--:--:--	--:--:--	--:--:--
1800										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1902	0	--:--:--	--:--:--	--:--:--
1906										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1843	0	--:--:--	--:--:--	--:--:--
1848										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1862	0	--:--:--	--:--:--	--:--:--
1860										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1806	0	--:--:--	--:--:--	--:--:--
1810										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1702	0	--:--:--	--:--:--	--:--:--
1704										

(external)\$

root@44ca1c6c0907:~\# **cli_script.sh 'stat lb vserver cpx-vip'**

exec: stat lb vserver cpx-vip

Virtual Server Summary

		vsvrIP	port	Protocol	State	Health
actSvcs						
cpx-vip		172.17.0.2	88	HTTP	UP	
100	2					

inactSvcs

cpx-vip	0
---------	---

Virtual Server Statistics

Rate (/s)	Total
-----------	-------

Vserver hits	0	101
Requests	0	101
Responses	0	101
Request bytes	0	8585
Response bytes	0	85850
Total Packets rcvd	0	708
Total Packets sent	0	408
Current client connections	--	0
Current Client Est connections	--	0
Current server connections	--	0
Current Persistence Sessions	--	0

Requests in surge queue	--	0
Requests in vserver's surgeQ	--	0
Requests in service's surgeQs	--	0
Spill Over Threshold	--	0
Spill Over Hits	--	0
Labeled Connection	--	0
Push Labeled Connection	--	0
Deferred Request	0	0
Invalid Request/Response	--	0
Invalid Request/Response Dropped	--	0
Vserver Down Backup Hits	--	0

```
Current Multipath TCP sessions          --          0

Current Multipath TCP subflows          --          0

Apdex for client response times.        --          1.00

Average client TTLB                     --          0

web1          172.17.0.3      80      HTTP      UP      51
0/s

web2          172.17.0.4      80      HTTP      UP      50
0/s

Done

root@44ca1c6c0907:~\#
```

NetScaler CPX 故障排除

本文档解释了如何排查您在使用 NetScaler CPX 时可能遇到的问题。通过本文档，您可以收集日志以确定原因，并针对与 NetScaler CPX 安装和配置相关的一些常见问题应用变通方法。

- 为什么 NetScaler CPX 中的内存使用率总是很高？

内存使用率是通过比较 NetScaler CPX 的当前内存使用量与 NetScaler CPX 从操作系统 (OS) 分配给自身的总内存来确定的。NetScaler CPX 首先从操作系统分配最少的必要内存用于启动和基本操作。最初，由于 NetScaler CPX 为其操作分配了最少的必要内存，因此内存消耗百分比很高。最终，随着 NetScaler CPX 操作规模的扩大，对更多资源的需求增加，NetScaler CPX 会从操作系统增量分配额外的内存，内存使用量也随之增加，从而导致内存使用率很高。

因此，由于 NetScaler CPX 的动态内存分配策略，NetScaler CPX 通常显示较高的内存使用率，这反映了它对资源的有效利用，而不是可用内存不足。

- 如何查看 NetScaler CPX 日志？

如果 NetScaler CPX 是使用 `tty:true` 选项部署的，您可以使用 `kubectl logs` 命令查看 NetScaler CPX 日志。您可以运行以下命令来显示日志：

```
kubectl logs <pod-name> [-c <container-name>] [-n <namespace-name>]
```

示例，

```
kubectl logs cpx-ingress1-69b9b8c648-t8bgn -c cpx -n citrix-adc
```

以下是使用 `tty:true` 选项部署 NetScaler CPX pod 的示例：

```
containers:

  - name: cpx-ingress

    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-58.30"

    tty: true
```

```
securityContext:

  privileged: true

env:
```

您可以在 NetScaler CPX 文件系统的 `/cpx/log/boot.log` 文件中找到更多启动日志。

注意：要获取 pod 名称，请运行 `kubectl get pods -o wide` 命令。

- 如何从 NetScaler CPX 收集技术支持包？

您可以在 Kubernetes 主节点的 shell 界面上运行以下命令以收集 NetScaler CPX 技术支持包：

```
kubectl exec <cpx-pod-name> [-c <cpx-container-name>] [-n
<namespace-name>] /var/netscaler/bins/cli_script.sh "show
techsupport"
```

您可以在 NetScaler CPX 文件系统的 `/var/tmp/support` 目录中查看技术支持包。使用 `scp` 或 `kubectl cp` 将技术支持包从 NetScaler CPX 复制到所需目标。

示例：

```
root@localhost# kubectl exec cpx-ingress1-55b9b6fc75-t5kc6 -c cpx -n
citrix-adc /var/netscaler/bins/cli_script.sh "show techsupport"

exec: show techsupport

Scope:  NODE

Done

root@localhost# kubectl cp cpx-ingress1-55b9b6fc75-t5kc6:var/tmp/
support/collector_P_192.168.29.232_31Aug2020_07_30.tar.gz /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz -c cpx

root@localhost# ll /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz
```

```
-rw-r--r-- 1 root root 1648109 Aug 31 13:23 /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz
```

- NetScaler CPX pod 为什么在启动时卡住？

您可以使用 `kubectl describe pods` 命令检查 pod 状态。运行以下命令以了解 pod 状态：

```
kubectl describe pods <pod-name> [-c <container-name>] [-n
<namespace-name>]
```

示例：

```
kubectl describe pods cpx-ingress1-69b9b8c648-t8bgn
```

如果 pod 事件显示容器已启动，则必须检查 pod 日志。

- 如何在 NetScaler CPX pod 和 Kubernetes 主节点之间复制文件？

建议使用 Docker 的卷挂载功能将 `/cpx` 目录挂载到主机的文件系统。如果 NetScaler CPX 容器退出并生成核心转储，则日志和其他重要数据可在挂载点上获取。

您可以使用以下任一命令在 NetScaler CPX pod 和 Kubernetes 主节点之间复制文件：

kubectl cp：您可以运行以下命令将文件从 pod 复制到节点：

```
kubectl cp <pod-name>:<absolute-src-path> <dst-path> [-c
<container-name>] [-n <namespace-name>]
```

示例：

```
root@localhost:~# kubectl cp cpx-ingress-596d56bb6-zbx6h:cpx/log/
boot.log /tmp/cpx-boot.log -c cpx-ingress

root@localhost:~# ll /tmp/cpx-boot.log

-rw-r--r-- 1 root root 7880 Sep 11 00:07 /tmp/cpx-boot.log
```

scp：您可以使用此命令在 NetScaler CPX pod 和 Kubernetes 节点之间复制文件。运行以下命令将文件从 pod 复制到节点。当提示输入密码时，提供 SSH 用户的密码：

```
scp <user>@<pod-ip>:<absolute-src-path> <dst-path>
```

示例：

```
root@localhost:~# scp nsroot@192.168.29.198:/cpx/log/boot.log /tmp/cpx-
boot.log

nsroot@192.168.29.198's password:

boot.log

100% 7880      5.1MB/s   00:00

root@localhost:~#
```

- 如何在 NetScaler CPX 上捕获数据包？

要在 NetScaler CPX 上捕获数据包，请使用 `kubectl exec` 命令启动 NetScaler CPX 的 shell 界面。运行以下命令以启动 NetScaler CPX pod 的 shell 界面：

```
kubectl exec -it pod-name [-c container-name] [-n namespace-name] bash
```

示例：

```
kubectl exec -it cpx-ingress1-69b9b8c648-t8bgn -c cpx -n citrix-adc
bash
```

然后，运行以下命令开始数据包捕获：

```
cli_script.sh "start nstrace -size 0"
```

如果您想停止正在进行的包捕获，请运行以下命令：

```
cli_script.sh "stop nstrace"
```

您可以在 NetScaler CPX 文件系统上的 `/cpx/nstrace/time-stamp` 目录中查看捕获到 `.cap` 文件中的数据包。

- 即使 NetScaler CPX 使用 `LS_IP=<ADM-IP>` 环境变量部署，为什么许可证服务器仍未配置？

确保可以从部署 NetScaler CPX 的节点访问许可证服务器。您可以使用 `ping <ADM-IP>` 命令验证从 NetScaler CPX 节点到 NetScaler Console 的连接。

如果可以从节点访问 NetScaler Console，则必须在 `/cpx/log/boot.log` 文件中验证许可证服务器配置日志。您还可以使用 NetScaler CPX pod 的 shell 界面上的以下命令检查许可证服务器配置：

```
cli_script.sh "show licenseserver"
```

示例：

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
licenseserver"

exec: show licenseserver

ServerName: 10.106.102.199Port: 27000 Status: 1 Grace: 0
Gptimeleft: 720

Done
```

- 即使在 NetScaler CPX 上成功配置许可证服务器后，为什么 NetScaler CPX 上仍未配置池化许可证？

在 `/cpx/log/boot.log` 文件中验证许可证配置日志。您还可以使用 NetScaler CPX pod 的 shell 界面上的以下命令验证 NetScaler CPX 上配置的池化许可证：

```
cli_script.sh "show capacity"
```

示例，

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
capacity"

exec: show capacity

Actualbandwidth: 1000 MaxVcpuCount: 2 Edition: Platinum Unit:
Mbps Bandwidth: 0` `Maxbandwidth: 40000 Minbandwidth: 20
Instancecount: 1
```

Done

此外，请确保所需的许可证文件已上传到许可证服务器。在 NetScaler CPX 上成功配置许可证服务器后，您还可以使用以下命令验证许可证服务器上可用的许可证。在 NetScaler CPX pod 的 shell 界面上运行该命令：

```
cli_script.sh "sh licenseserverpool"
```

示例：

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
licenseserverpool"
```

```
exec: show licenseserverpool
```

Instance Total	: 5
Instance Available	: 4
Standard Bandwidth Total	: 0 Mbps
Standard Bandwidth Availabe	: 0 Mbps
Enterprise Bandwidth Total	: 0 Mbps
Enterprise Bandwidth Available	: 0 Mbps
Platinum Bandwidth Total	: 10.00 Gbps
Platinum Bandwidth Available	: 9.99 Gbps
CP1000 Instance Total	: 100
CP1000 Instance Available	: 100

Done

- 为什么 NITRO API 调用会从 NetScaler CPX 收到 *Connection Refused* 响应？

从 NetScaler CPX 12.1 版本开始，NITRO API 的默认端口是 9080（不安全）和 9443（安全）。确保您尝试访问的 NetScaler CPX 的 NITRO 端口在 pod 上已公开。您可以运行 `kubectl describe` 命令在 NetScaler CPX 容器部分查看 NetScaler CPX 容器的公开和映射端口：

```
kubectl describe pods <pod-name> | grep -i port
```

示例：

```
ng472 | grep -i port
```

```
Ports:          80/TCP, 443/TCP, 9080/TCP, 9443/TCP
```

```
Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP
```

```
NS_HTTP_PORT:   9080
```

```
NS_HTTPS_PORT:  9443
```

```
Port:           <none>
```

```
Host Port:      <none>
```

```
NS_PORT:        80
```

- 即使没有或只有少量流量，为什么 NetScaler CPX 中的 NSPPE 进程仍会占用大部分 CPU 使用率？

如果 NetScaler CPX 使用 `CPX_CONFIG='{ "YIELD": "NO" }'` 环境变量部署，即使没有或只有少量流量，NSPPE 进程也会占用 100% 的 CPU 使用率。如果您不希望 NSPPE 进程占用 CPU 使用率，则必须在不使用 `CPX_CONFIG='{ "YIELD": "NO" }'` 环境变量的情况下部署 NetScaler CPX。默认情况下，CPX 中的 NSPPE 进程配置为不占用或消耗 CPU 使用率。

- 为什么 NetScaler CPX 即使已使用注册到 NetScaler Console 所需的环境变量进行部署，也未在 NetScaler Console 中列出？

您可以在 NetScaler CPX 文件系统上的 `/cpx/log/boot.log` 文件中找到 NetScaler CPX 注册到 NetScaler Console 的日志。

您可以使用 `ping` 命令从 NetScaler CPX pod 验证 NetScaler Console IP 地址的可访问性。此外，请确保为 NetScaler CPX 容器配置了所有必需的 NetScaler Console 注册环境变量。

- `NS_MGMT_SERVER` : 指定 ADM IP 地址或 FQDN。
 - `HOST` : 指定节点 IP 地址。
 - `NS_HTTP_PORT` : 指定节点上映射的 HTTP 端口。
 - `NS_HTTPS_PORT` : 指定节点上映射的 HTTPS 端口。
 - `NS_SSH_PORT` : 指定节点上映射的 SSH 端口。
 - `NS_SNMP_PORT` : 指定节点上映射的 SNMP 端口。
 - `NS_ROUTABLE` : NetScaler CPX pod IP 地址无法从外部路由。
 - `NS_MGMT_USER` : 指定 ADM 用户名。
 - `NS_MGMT_PASS` : 指定 ADM 密码。
- 更改 nsroot 用户的密码后, 为什么 `cli_script.sh` 会显示 *Invalid user name or password* 错误消息?

命令 `cli_script.sh` 是 NetScaler CPX 上 NSCLI 的包装实用程序。它将第一个参数作为命令字符串或文件路径运行, 第二个参数是可选的凭据。如果 nsroot 用户的密码已更改, 则需要将凭据作为第二个参数提供给 `cli_script.sh`。您可以运行以下命令以使用凭据运行 NSCLI :

```
cli_script.sh "<command>" ":<username>:<password>"
```

示例 :

```
root@087a1e34642d:/# cli_script.sh "show ns ip"

exec: show ns ip

ERROR: Invalid username or password

root@087a1e34642d:/# cli_script.sh "show ns ip" ":nsroot:nsroot123"

exec: show ns ip
```

```

Ipaddress      Traffic Domain      Type      Mode      Arp
Icmp    Vserver  State
-----
-----

172.17.0.3      0      NetScaler IP      Active      Enabled
Enabled  NA      Enabled

192.0.0.1      0      SNIP      Active      Enabled
Enabled  NA      Enabled

Done

```

- 为什么使用 `root` 和 `nsroot` 用户 SSH 到 NetScaler CPX 会失败？

从 13.0-64.35 版本开始，NetScaler CPX 会为 SSH 用户（`root` 和 `nsroot`）生成并更新默认密码。如果您尚未手动更改密码，则可以在 NetScaler CPX 文件系统上的 `/var/deviceinfo/random_id` 中找到 SSH 用户的密码。

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.