



Citrix ADC CPX 13.0

Contents

关于 NetScaler CPX	2
体系结构和通信流	4
NetScaler CPX 许可	7
在 Docker 中部署 NetScaler CPX 实例	14
将 NetScaler CPX 实例添加到 NetScaler ADM	21
配置 NetScaler CPX	23
在 NetScaler CPX 实例上配置 AppFlow	26
使用配置文件配置 NetScaler CPX	29
NetScaler CPX 中的动态路由支持	30
为 NetScaler CPX 配置高可用性	33
配置 Docker 日志记录驱动程序	38
升级 NetScaler CPX 实例	39
在 NetScaler CPX 实例中使用通配符虚拟服务器	41
将 NetScaler CPX 部署为代理以支持东西通信流	41
在单主机网络中部署 NetScaler CPX	45
在多主机网络中部署 NetScaler CPX	46
为 NetScaler CPX 部署对网络的直接访问	51
使用 ConfigMap 在 Kubernetes 中配置 NetScaler CPX	52
部署 NetScaler CPX 作为 Kubernetes 节点的本地 DNS 缓存	54
在 Google Compute Engine 上部署 NetScaler CPX 代理	58
NetScaler CPX 故障排除	78

关于 NetScaler CPX

November 23, 2023

NetScaler CPX 是一个基于容器的应用程序交付控制器，可以在 Docker 主机上进行配置。NetScaler CPX 使客户能够利用 Docker 引擎功能，将 NetScaler ADC 负载平衡和流量管理功能用于基于容器的应用程序。可以将一个或多个 NetScaler CPX 实例作为独立的实例在 Docker 主机上进行部署。

一个 NetScaler CPX 实例提供高达 1 Gbps 的吞吐量。

作为 NetScaler ADC 的容器化外形，NetScaler CPX 可以很好地集成到 Kubernetes 环境中，是 Citrix 云原生解决方案不可分割的一部分。Citrix 云原生解决方案可帮助您在 Kubernetes 环境中快速、敏捷、高效地创建和交付软件应用程序。使用 Citrix 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。

有关详细信息，请参阅 [Citrix 云原生解决方案](#)。

本文档假定您熟悉 Docker 及其工作方式。有关 Docker 的信息，请参阅 Docker 文档，网址为 <https://docs.docker.com>。

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 内容交换
 - SSL 卸载
 - IPv6 协议转换
 - Microsoft SQL, MySQL 负载平衡
 - AppExpert 速率控制
 - 订户感知的流量定向
 - 动态路由协议
 - 浪涌保护和优先级队列
- 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 重写和响应程序
 - L4 DoS 防护

- L7 DoS 防护
- Web Application Firewall (WAF)。NetScaler CPX 支持其他 NetScaler ADC 外形规格支持的所有 WAF 功能。有关支持的 WAF 功能的信息，请参阅 [Citrix Web 应用程序防火墙](#)。
- 应用程序流量的身份验证、授权和审核 (AAA)
- TCP 协议优化
 - 多路径 TCP
 - Binary Increase Congestion Control (BIC) 和 cubic TCP
- 简单可管理性
 - 网络日志记录
 - AppFlow
 - NetScaler Application Delivery Management
 - 操作分析
- 应用程序优化
 - 集成缓存
- BGP 路由和路由运行状况注入 (RHI)
- 高可用性（包括第 2 层和第 3 层）

注意：

分配给 NetScaler CPX 设备的接口（Linux 主机）的 Rx、Tx、GRO、GSO 和 LRO 等接口功能被禁用。即使在 NetScaler CPX 设备停止之后，这些功能仍处于禁用状态。此外，此类接口的 MTU 更改为 1500 个字节。

支持的平台

以下平台支持 NetScaler CPX：

- Docker 版本 1.12 及更高版本
- Kubernetes
- Red Hat OpenShift
- 公有云
 - Amazon Elastic Kubernetes Service (EKS)
 - Azure Kubernetes Service (AKS)
 - Google Kubernetes Engine (GKE)
- Rancher
- Pivotal Container Service (PKS)

体系结构和通信流

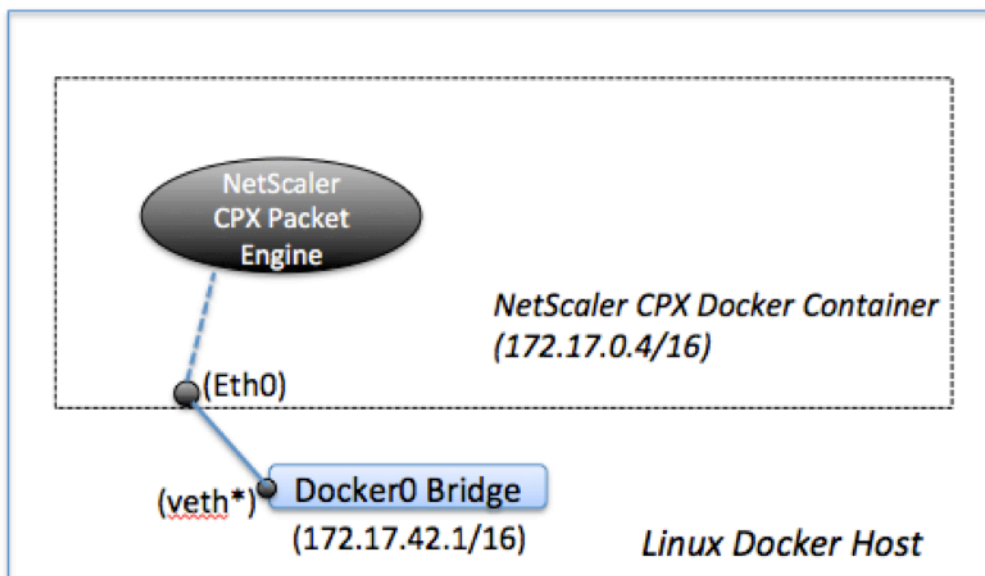
November 23, 2023

本节介绍了 NetScaler CPX 桥接模式架构和流量。NetScaler CPX 也可以在主机模式下部署。

在 Docker 主机上置备 NetScaler CPX 实例时，Docker 引擎会为 CPX 实例创建虚拟接口 eth0。此 eth0 接口直接连接至 docker0 桥接上的虚拟接口 (veth*)。Docker 引擎还为网络 172.17.0.0/16 中的 NetScaler CPX 实例分配 IP 地址。

CPX 实例的默认网关是 docker0 桥接的 IP 地址，这意味着与 NetScaler CPX 实例的任何通信都是通过 Docker 网络进行的。从 docker0 桥接接收的所有传入流量都是由 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 主机上的 NetScaler CPX 实例的体系结构。



如何在 **NetScaler CPX** 上使用单一 IP 地址

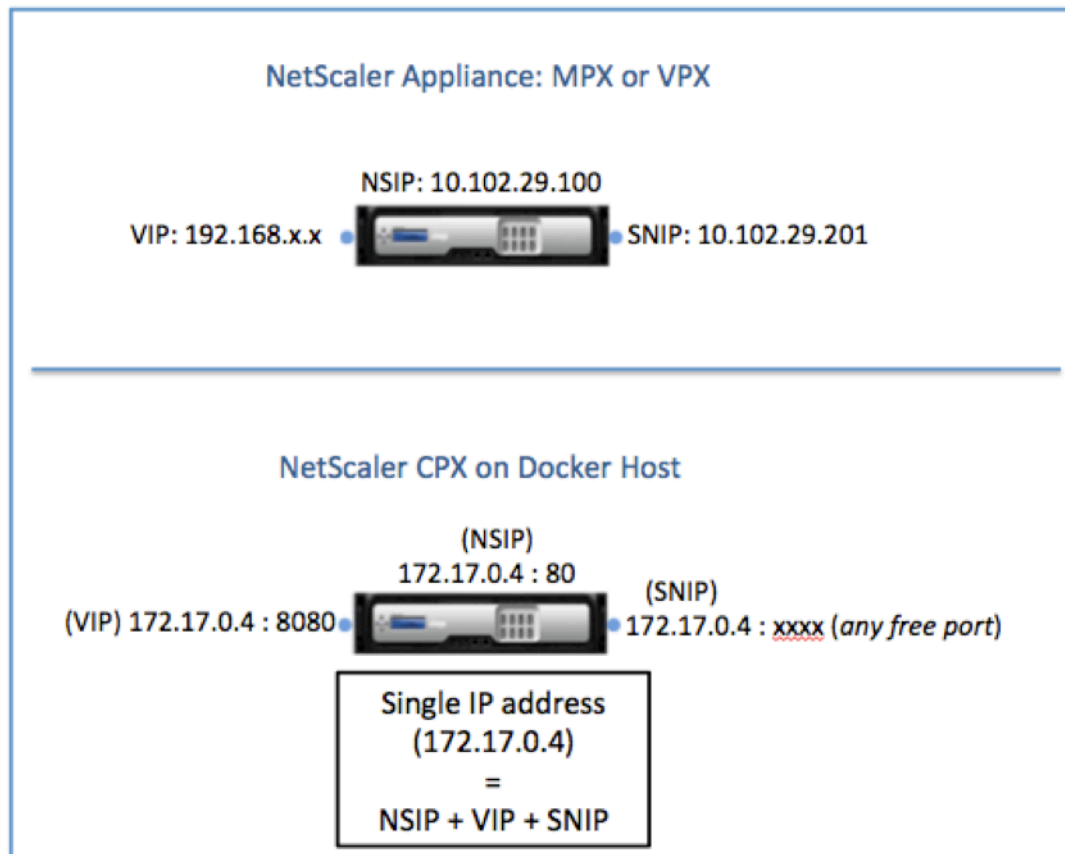
常规 NetScaler MPX 或 VPX 设备至少需要三个 IP 地址才能正常工作：

- 名为 NetScaler ADC IP (NSIP) 地址的管理 IP 地址
- 子网 IP (SNIP) 地址，用于与服务器场通信
- 虚拟服务器 IP (VIP) 地址，用于接受客户端请求

NetScaler CPX 实例使用一个单一 IP 地址，用于管理以及数据流量。

置备时，Docker 引擎只为一个 NetScaler CPX 实例分配一个专用 IP 地址（单一 IP 地址）。NetScaler ADC 实例的三个 IP 功能会多路复用到一个 IP 地址。此单一 IP 地址使用不同的端口号来执行 NSIP、SNIP 和 VIP 功能。

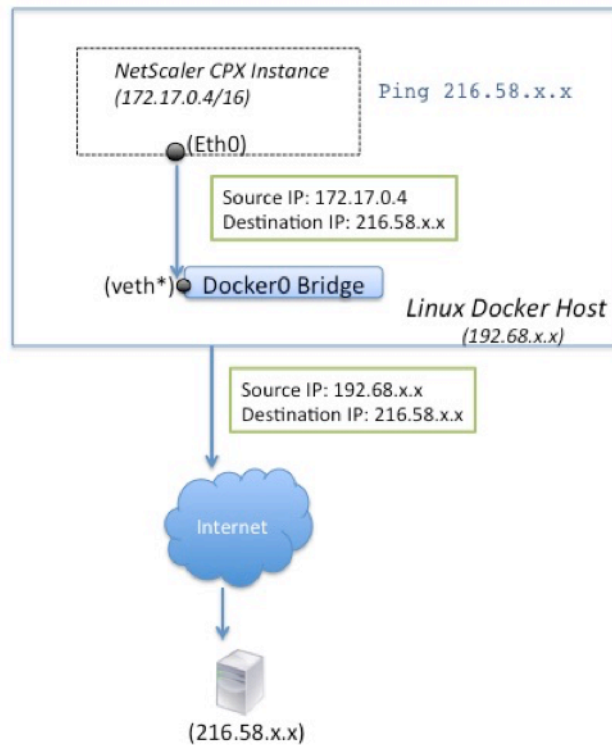
下图说明如何使用单一 IP 地址来执行 NSIP、SNIP 和 VIP 的功能。



源自 **NetScaler CPX** 实例的请求的通信流

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



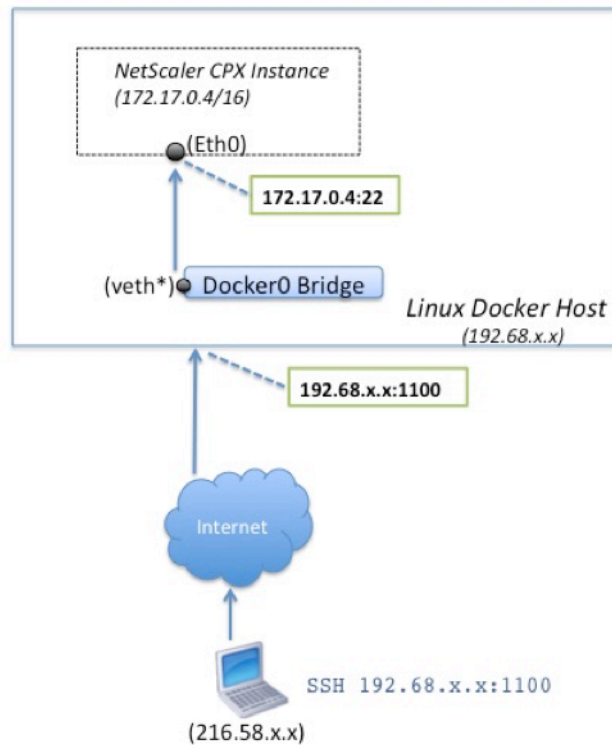
在此示例中，由 eth0 接口上的数据包引擎发送 ping 请求，来源 IP 地址为 NetScaler CPX IP 地址 (172.17.0.4)。然后 Docker 主机执行网络地址转换 (NAT) 以将主机 IP 地址 (192.68.x.x) 添加为来源 IP 地址，并将请求发送至目标 (216.58.x.x)。反之，来自目标 IP 地址的响应使用相同的路径。Docker 主机对响应执行 NAT 并将响应转发至 NetScaler CPX 实例的 eth0 接口上。

源自外部网络的请求的通信流

为了能够进行外部通信，置备 NetScaler CPX 时，必须设置一些参数，以便 Docker 可以公开某些端口（例如 80、22 以及您想要的任何其他端口）。如果您在置备期间未设置要公开的任何端口，那么必须在 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，然后该主机执行端口地址转换 (PAT) 以将公用 IP 地址和端口映射至 NetScaler CPX 实例的单一 IP 地址和端口，并将流量转发至实例。

下图显示了 Docker 主机如何执行端口地址转换以将流量导向至 NetScaler CPX 单一 IP 地址和端口。



在此示例中，Docker 主机 IP 地址是 192.68.x.x，NetScaler CPX 实例的单一 IP 地址是 172.17.0.4。NetScaler CPX 实例的 SSH 端口 22 映射至 Docker 主机上的端口 1100。来自客户端的 SSH 请求通过 IP 地址 192.68.x.x 上的端口 1100 接收。Docker 主机执行端口地址转换以将此地址和端口映射至单一 IP 地址 172.17.0.4 上的端口 22，并转发客户端请求。

NetScaler CPX 许可

March 25, 2024

[NetScaler CPX](#) 是一个基于容器的应用程序交付控制器，可以在 Docker 主机上进行配置，以对基于微服务的应用程序进行负载平衡。为了提高应用程序交付性能，您需要获得许可的 CPX。NetScaler CPX 支持池许可。NetScaler ADM 可以充当您的许可证服务器，为您的 NetScaler CPX 实例提供许可。

NetScaler ADM 既可在本地提供，也可以作为云服务提供。可以使用 Citrix ADM 管理所有 Citrix ADC 外形规格的池容量许可证。

有关本地 Citrix ADM 的信息，请参阅[本地 Citrix ADM](#)。有关 NetScaler ADM 服务的信息，请参阅[NetScaler ADM 服务](#)。

NetScaler CPX 许可的类型

NetScaler CPX 支持本地和云端部署的带宽和虚拟 CPU（核心）池许可。

带宽池：NetScaler CPX 许可证可以根据实例的带宽消耗进行分配。可以使用池许可来最大限度地提高带宽利用率，方法是确保对实例进行必要的带宽分配而不超过其要求。目前，NetScaler CPX 仅支持高级带宽池许可。有关带宽池的详细信息，请参阅[基于带宽的许可](#)。

vCPU 池：在基于虚拟 CPU 使用量的许可中，许可证指定了特定 NetScaler CPX 实例有权获得的 CPU 数量。因此，NetScaler CPX 只能从许可证服务器中查看虚拟 CPU 数量的许可证。NetScaler CPX 根据系统中运行的 CPU 数量来检查许可证。有关 vCPU 池的详细信息，请参阅[NetScaler ADC 虚拟 CPU 许可](#)。

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	最低带宽单位
NetScaler CPX	40000 注意：这取决于 CPU 频率、生成等。	20 Mbps	1	16	10 Mbps

注意：Citrix 目前正在为基于云的公共产品开发基于消费的 NetScaler CPX 或基于“按增长付费”的许可模式。准备就绪后，它将在公有云市场上提供以供使用。

NetScaler CPX 许可的工作原理

NetScaler CPX 池容量：一个通用许可证池，您的 NetScaler CPX 实例可以从中检出一个实例许可证，并且只能查看所需的带宽。当实例不再需要这些资源时，就会将其重新签入公用池，以使资源可用于需要这些许可证的其他实例。

NetScaler CPX 签入和签出许可：NetScaler ADM 按需分配许可证 NetScaler CPX 实例。配置 NetScaler CPX 实例后，NetScaler CPX 实例可以从 NetScaler ADM 签出许可证，并在实例销毁时将其许可证退回 NetScaler ADM。

NetScaler CPX 行为：单个 NetScaler CPX 实例签出吞吐量高达 1 Gbps，仅从实例池签出，不能从带宽许可证池中签出。NetScaler CPX 以这种方式运行，带宽利用率最高可达 1 Gbps。例如，如果 CPX 实例消耗 200 Mbps 的带宽，它将使用许可证的实例池，而非带宽池。但是，如果 NetScaler CPX 实例消耗 1200 Mbps 的吞吐量，则使用实例池中的前 1000 Mbps 吞吐量，其余 200 Mbps 将从带宽池中消耗。

NetScaler CPX Express

NetScaler CPX Express 是一个免费软件版本，适用于本地和云部署。当您从 [Quay](#) 存储库下载 NetScaler CPX 实例时，这是不需要许可证文件的 POC 可用的默认容量，它具有以下功能：

- 20 Mbps 带宽
- 最多 250 个 SSL 会话
- 20 Mbps SSL 吞吐量

您必须许可您的 NetScaler CPX 实例才能升级，以获得更好的性能和生产部署。

NetScaler CPX 许可模式

Citrix 为 NetScaler CPX 提供了一系列产品许可模式，以满足您组织的需求。您可以选择 vCPU 或带宽以及本地或云等选项。

可以选择以下任一模式，具体取决于您的要求：

- 通过 ADM 服务获得 NetScaler CPX 的基于带宽的许可
- 通过 ADM 服务获得 NetScaler CPX 的基于 vCPU 的许可
- 从本地 ADM 获得 NetScaler CPX 的基于带宽的许可
- 从本地 ADM 获得 NetScaler CPX 的基于 vCPU 的许可

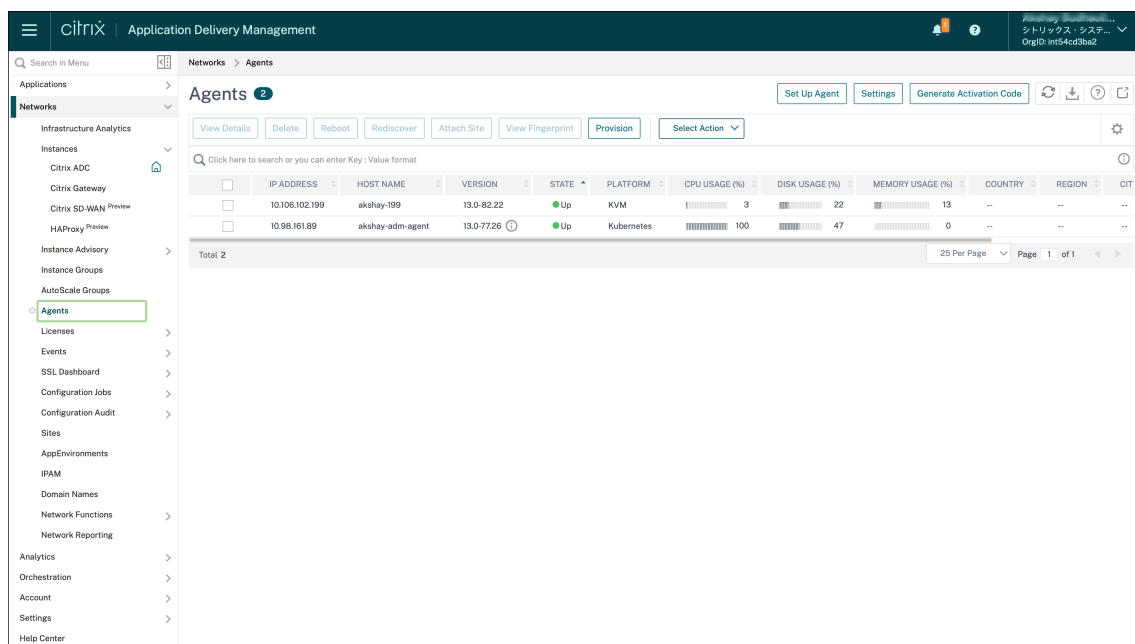
从 **NetScaler ADM** 服务为 **NetScaler CPX** 提供基于带宽和基于 **vCPU** 的许可

执行以下步骤，为来自 NetScaler ADM 服务的 NetScaler CPX 提供基于带宽的许可证和基于 vCPU 的许可证。

1. 设置 NetScaler ADM。

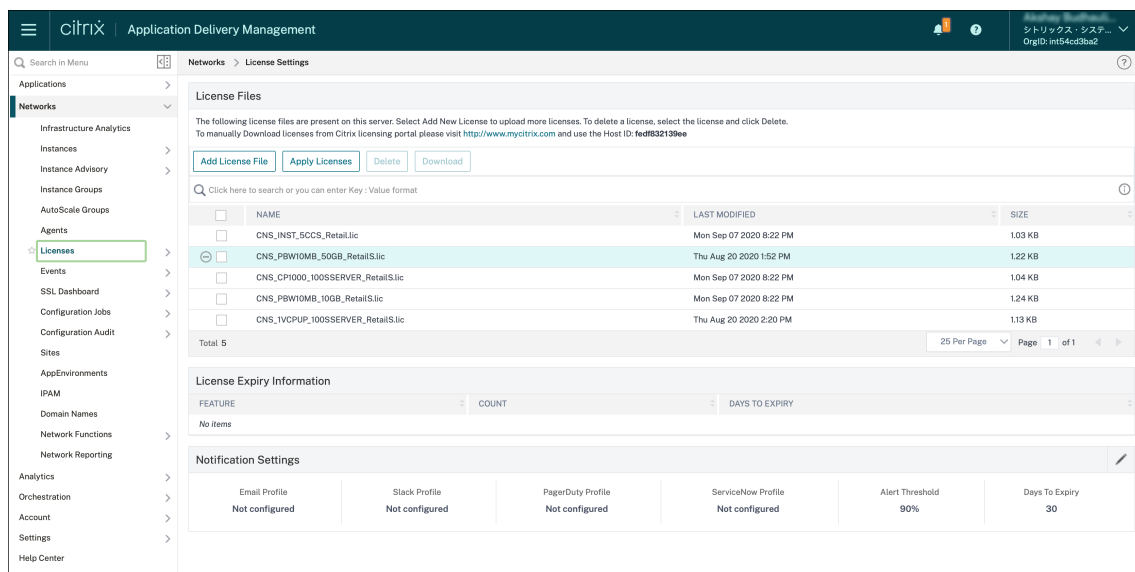
确保 NetScaler ADM 服务设置在 NetScaler ADM 代理下正常运行。您必须拥有 NetScaler ADM 服务和 NetScaler ADM 代理帐户，NetScaler CPX 许可才能正常运行。有关设置 NetScaler ADM 服务和 NetScaler ADM 代理的信息，请参阅 [NetScaler ADM 服务](#)。

注意：在此过程中，使用虚拟机管理程序（本地）NetScaler ADM 代理设置。在下图中，10.106.102.199 是用于许可 NetScaler CPX 的本地代理。



2. 将 Citrix ADC 实例许可证池添加到 Citrix ADM 服务。

假设您有一个可用于 ADM 服务的带宽许可证池。有关将许可证文件上传到 Citrix ADM 的信息，请参阅[配置池容量](#)。在下图中，`CNS_INST_200CC_Retail.lic` 用作带宽和实例许可证池。



3. 在 Kubernetes 群集中部署 NetScaler CPX 实例。确保将以下环境变量添加到 NetScaler CPX YAML 文件中，以许可 NetScaler CPX 实例。

对于来自 NetScaler ADM 服务的基于带宽的许可，请在 YAML 文件中指定以下环境变量：

- 名称：LS_IP
值：10.105.158.166 //步骤 1 中提到的 ADM 代理 IP

- 名称: LS_PORT
值: 27000 // ADM 许可证服务器进行侦听时所在的端口
- 名称: BANDWIDTH
值: 3000 //希望分配给 CPX 的容量 (Mbps)
- 名称: EDITION
值: Standard 或 Enterprise //用于选择包含 Standard、Platinum 和 Enterprise 的特定许可证版本。默认选择 Platinum。

对于来自 Citrix ADM 服务的基于 vCPU 的许可, 请在 YAML 文件中指定以下环境变量:

- 名称: LS_IP
值: 10.102.216.173//步骤 1 中提到的 ADM 代理 IP
- 名称: LS_PORT
值: 27000 // ADM 许可证服务器进行侦听时所在的端口
- 名称: CPX_CORES
值: 4 // 您希望分配的内核数量
- 名称: PLATFORM
值: CP1000 // 核心数。签出计数等于核心数。

4. 使用以下命令下载 `cpx-bandwidth-license-adm-service.yaml` 文件:

```
1 kubectl create namespace bandwidth
2 wget https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/cpx-licensing/manifest/cpx-bandwidth-license-adm-service.yaml
```

5. 使用以下命令在 Kubernetes 群集中部署编辑过的 YAML:

```
1 kubectl create -f cpx-bandwidth-license-adm-service.yaml -n bandwidth
```

6. 使用以下命令登录 NetScaler CPX 以验证实例化信息:

```
1 kubectl exec -it 'cpx-pod-ip-name' bash -n bandwidth
```

7. 要查看给定 NetScaler CPX 实例的许可信息, 请运行以下命令:

```
1 cli_script.sh "show licenseserver"
2 cli_script.sh "show capacity"
```

可以在 ADM 服务门户中跟踪分配的带宽和 vCPU 容量。

在本地 **NetScaler ADM** 为 **NetScaler CPX** 提供基于带宽的许可和基于 **vCPU** 的许可

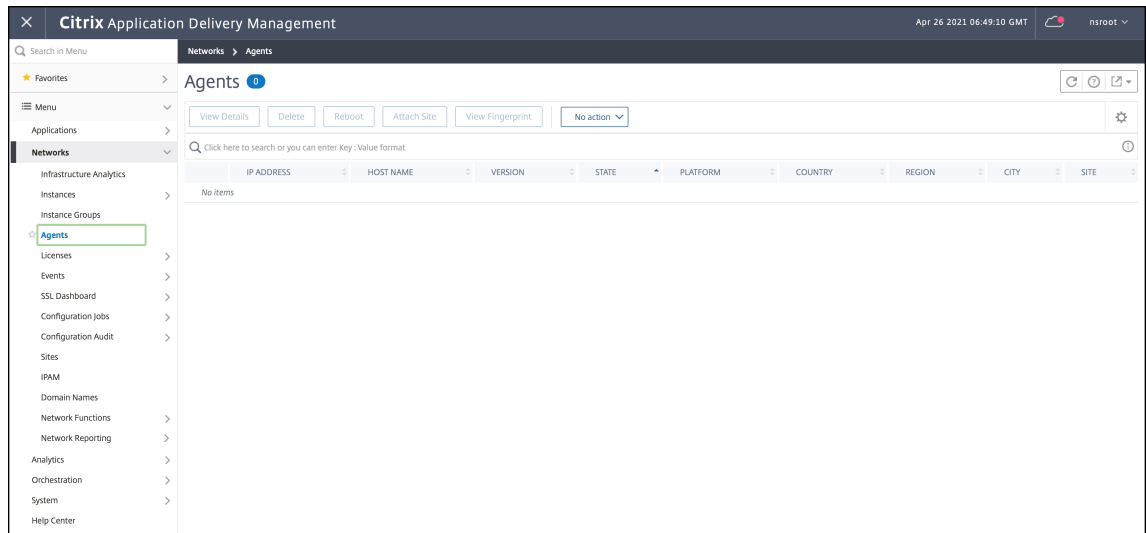
执行以下步骤, 从本地的 NetScaler ADM 向 NetScaler CPX 预置基于带宽和基于 vCPU 的配置。

1. 设置 NetScaler ADM。

确保本地 ADM 设置已准备就绪。无论是否部署了用于 NetScaler CPX 许可的 ADM 代理，都要确保本地的 NetScaler ADM 能够正常运行。

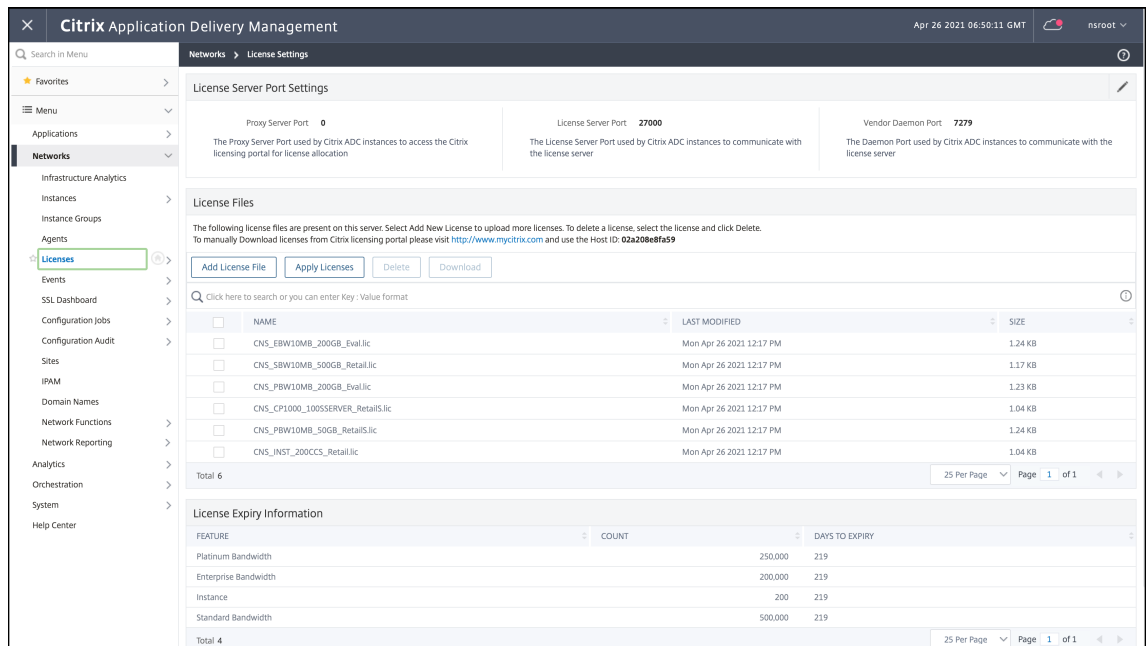
有关在本地设置 NetScaler ADM 和 NetScaler ADM 代理的信息，请参阅 [NetScaler ADM 服务](#)。

注意：在本示例中，使用带本地 ADM 的内置 ADM 代理。在下图中，您可以看到未部署代理。

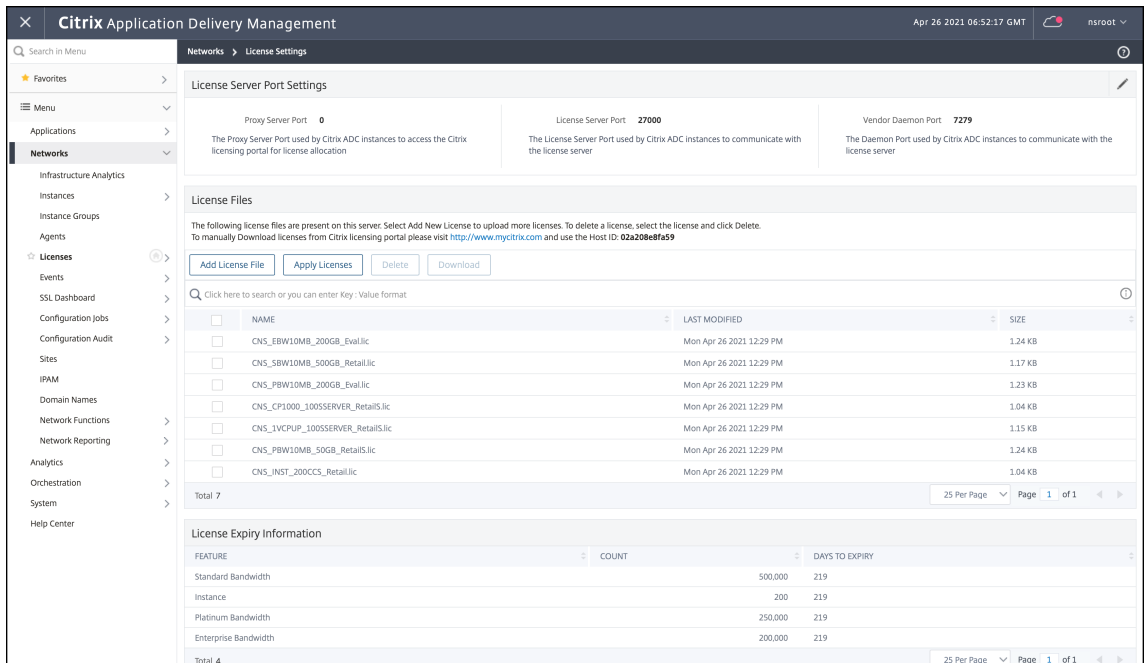


2. 将 Citrix ADC 实例许可证池添加到本地 ADM。

假设您有一个可用于本地 ADM 的带宽许可证池。要了解如何将许可证文件上传到 Citrix ADM，请参阅[许可](#)。在下图中，`CNS_INST_200CC_Retail.lic` 用作带宽和实例许可证池。



在下图中，CP1000 用作 vCPU 许可证池。



3. 在 Kubernetes 群集中部署 NetScaler CPX 实例。确保将以下环境变量添加到 NetScaler CPX YAML 文件中，以许可 NetScaler CPX 实例。

对于来自本地 NetScaler ADM 的基于带宽的许可，请在 YAML 文件中指定以下环境变量：

- 名称: LS_IP
值: 10.105.158.144 // ADM 本地实例 IP，如果您已部署 ADM 代理，那么这是步骤 1 中所述的代理 IP 地址
- 名称: LS_PORT
值: 27000 // ADM 许可证服务器进行侦听时所在的端口
- 名称: BANDWIDTH
值: 3000 // 希望分配给 CPX 的容量 (Mbps)

对于来自本地 Citrix ADM 的基于 vCPU 的许可，请在 YAML 文件中指定以下环境变量：

- 名称: LS_IP
值: 10.105.158.144 // ADM 本地实例 IP，如果您有 ADM 代理部署，那么这将是步骤 1 中所述的代理 IP
- 名称: LS_PORT
值: 27000 // ADM 许可证服务器进行侦听时所在的端口
- 名称: CPX_CORES
值: 4 // 您想分配的内核数
- 名称: PLATFORM
值: CP1000 // 核心数。签发计数等于核心数。

4. 使用以下命令下载 `cpx-bandwidth-license-adm-onprem.yaml` 文件：

```
1 kubectl create namespace bandwidth
2 wget https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/cpx-licensing/manifest/cpx-bandwidth-license-adm-onprem.yaml
```

5. 使用以下命令在 Kubernetes 群集中部署编辑过的 YAML:

```
1 kubectl create -f cpx-bandwidth-license-adm-onprem.yaml -n bandwidth
```

6. 使用以下命令登录 NetScaler CPX 以验证实例化信息:

```
1 kubectl exec -it <cpx-pod-ip-name> bash -n bandwidth
```

7. 要查看 NetScaler CPX 实例的许可信息, 请运行以下命令:

```
1 cli_script.sh "show licenseserver"
2 cli_script.sh "show capacity"
```

可以在 ADM 本地门户中跟踪分配的带宽和 vCPU 容量。

用于清理部署的命令

可以使用以下命令清理各种 YAML 部署:

```
1 kubectl delete -f cpx-bandwidth-license-adm-service.yaml -n bandwidth
2 kubectl delete -f cpx-core-license-adm-service.yaml -n core
3 kubectl delete -f cpx-bandwidth-license-adm-onprem.yaml -n bandwidth
4 kubectl delete -f cpx-core-license-adm-onprem.yaml -n core
5 kubectl delete namespace bandwidth
6 kubectl delete namespace core
```

在 Docker 中部署 NetScaler CPX 实例

November 23, 2023

NetScaler CPX 实例在 Quay 容器注册表中作为 Docker 映像文件提供。要部署实例, 请从 Quay 容器注册表下载 NetScaler CPX 映像, 然后使用 `docker run` 命令或 Docker 撰写工具部署实例。

必备条件

请确保:

- Docker 主机系统至少具有:

- 1 个 CPU
- 2 GB RAM

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。对于您添加的每个附加处理引擎，请确保 Docker 主机包含相应数量的 vCPU 和内存量 (GB)。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。

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

注意：NetScaler CPX 在 ubuntu 版本 16.04.5、内核版本 4.4.0-131-generic 上运行时出现问题。因此，不建议在 ubuntu 版本 16.04.5 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，不建议在这些版本中使用 Citrix NetScaler CPX：

- 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：

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

例如，如果要下载版本 13.0-64.35，请使用以下命令：

```
1 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：

```
1 root@ubuntu:~# docker images | grep 'citrix-k8s-cpx-ingress'
2 quay.io/citrix/citrix-k8s-cpx-ingress          13.0-64.35
          952a04e73101                2 months ago          469 MB
```

您可以从 Quay 容器注册表中部署最新的 NetScaler CPX 映像。

注意：建议使用来自 Quay 容器注册表的最新 NetScaler CPX 映像来使用 NetScaler CPX 的最新功能。

使用 **docker run** 命令部署 **NetScaler CPX** 实例

在主机上，可以使用您加载到主机中的 NetScaler CPX Docker 映像，在 Docker 容器中安装 NetScaler CPX 实例。使用 `docker run` 命令，采用默认 NetScaler CPX 配置安装 NetScaler CPX 实例。

重要：

如果您从 <https://www.citrix.com/products/netscaler-adc/cpx-express.html> 中下载了 NetScaler CPX Express，请务必阅读并理解可在 <https://www.citrix.com/products/netscaler-adc/cpx-express.html> 获得的最终用户许可协议 (EULA)，并在部署 NetScaler CPX 实例时接受最终用户许可协议。

使用以下 **docker run** 命令在 Docker 容器中安装 NetScaler CPX 实例：

```
1 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=' {
2   "YIELD" : "NO" }
3   ' -e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT> -e PLATFORM=CP1000 -v
   <host_dir>:/cpx <REPOSITORY>:<TAG>
4 <!--NeedCopy-->
```

```
1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
   CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e CPX_CONFIG='{
2   "YIELD": "No" }
3   ' -e LS_IP=10.102.38.134 -e PLATFORM=CP1000 -v /var/cpx:/cpx --name
   cpx_host cpx:13.0-x.x
4 <!--NeedCopy-->
```

此示例创建了一个基于 NetScaler CPX Docker 映像的名为 `mycpx` 的容器。

`-P` 参数是必需的。它让 Docker 映射 NetScaler CPX Docker 映像中的端口。这意味着将端口 9080、22、9443 和 161/UDP 映射到从用户定义的范围中随机选择的 Docker 主机上的端口。完成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，并在每次启动或重新启动容器时设置。端口使用情况如下：

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

如果您需要静态端口映射，请使用 `-p` 参数手动设置它们。

`--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 提供所有系统权限。如果您想在桥接模式下使用单核或多核运行 NetScaler CPX，则可以使用 `--cap-add=NET_ADMIN` 选项代替此选项。`--cap-add=NET_ADMIN` 选项使您能够以完全的网络权限运行 NetScaler CPX 容器。

**`--net=host` 是标准 `docker run` 命令选项，用于指定容器在主机网络堆栈中运行，并有权访问所有网络设备。

注意

如果您在网桥或不在网络中运行 NetScaler CPX，请忽略此选项。

`-e NS_NETMODE="HOST"` 是 NetScaler CPX 特定的环境变量，允许您指定 NetScaler CPX 在主机模式下启动。NetScaler CPX 在主机模式下启动后，它会在主机上配置 4 条默认 iptables 规则，用于管理对 NetScaler CPX 的访问权限。它使用以下端口：

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

如果您要指定其他端口，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

如果您在网桥或不在网络中运行 NetScaler CPX，请忽略此环境变量。

`-e CPX_CORES` 是可选的 NetScaler CPX 特定的环境变量。您可以使用它来提高 NetScaler CPX 实例的性能，方法是定义希望 NetScaler CPX 容器启动的处理引擎数。

注意：NetScaler CPX 可以支持 1 到 16 个内核。

注意

对于您添加的每个附加处理引擎，请确保 Docker 主机包含相应数量的 vCPU 和内存量 (GB)。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。

`-e EULA = yes` 是必需的 NetScaler CPX 特定的环境变量，这是验证您已阅读并理解此处提供的最终用户许可协议 (EULA) 所必需的：<https://www.citrix.com/products/netscaler-adc/cpx-express.html>。

`-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，则可以使用 `-e CPX_NW_DEV` 环境变量为 NetScaler CPX 容器分配专用的网络接口。您需要定义以空格分隔的网络接口。在您卸载 NetScaler CPX 容器之前，您定义的网络接口由 NetScaler CPX 容器保存。配置 NetScaler CPX 容器后，所有分配的网络接口都将添加到 NetScaler ADC 网络命名空间。

注意

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

```

1 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
2 <!--NeedCopy-->

```

`-e CPX_CONFIG` 是 NetScaler CPX 特定的环境变量，通过它可以控制 NetScaler CPX 容器的吞吐量性能。NetScaler CPX 不接收任何传入流量进行处理时，它会在此闲置时间期间让出 CPU，从而导致吞吐量性能较低。在这种情况下，您可以使用 `CPX_CONFIG` 环境变量控制 NetScaler CPX 容器的吞吐量性能。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 容器在闲置情况下让出 CPU，请定义 { "YIELD" : "Yes" }
- 如果您希望 NetScaler CPX 容器避免在闲置情况下让出 CPU，以便获得高吞吐量性能，请定义 { "YIELD" : "No" }

```

1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
  EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{
2   "YIELD":"No" }
3   ' -e PLATFORM=CP1000 --name cpx_host cpx:13.0-x.x
4 <!--NeedCopy-->

```

```

1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
  EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{
2   "YIELD":"Yes" }
3   ' -e PLATFORM=CP1000 --name cpx_host cpx:13.0-x.x
4 <!--NeedCopy-->

```

`-v` 参数是可选参数，用于指定 NetScaler CPX 装载目录 `/cpx` 的装载点。装载点是主机上的一个目录，在此装载 `/cpx` 目录。`/cpx` 目录存储日志、配置文件、SSL 证书和内核转储文件。在该示例中，装载点是 `/var/cpx`，NetScaler CPX 装载目录是 `/cpx`。

如果您购买了许可证或具有评估版许可证，则可以将许可证上载到许可证服务器，并使用 `docker run` 命令指定许可证服务器位置，方法是使用 `-e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT>` 参数。在此情况下，不必接受 EULA。

```

1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
  CPX_CORES=5 -e CPX_CONFIG='{
2   "YIELD":"No" }
3   ' -e LS_IP=10.102.38.134 -e PLATFORM=CP1000 --name cpx_host cpx:13.0-x
  .x
4 <!--NeedCopy-->

```

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。

可以使用以下命令来查看您的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** 命令部署更轻版本的 **NetScaler CPX**

Citrix 提供了 NetScaler CPX 的较轻版本，它消耗的运行时内存更少。NetScaler CPX 的较轻版本可以在服务网格部署中作为副车部署。

NetScaler CPX 的较轻版本支持以下功能：

- 应用程序可用性
 - L4 负载平衡和 L7 内容交换
 - SSL 卸载
 - IPv6 协议转换
- 应用程序安全性
 - L7 重写和响应程序
- 简单可管理性
 - 网络日志记录
 - AppFlow

要实例化更轻的 NetScaler CPX 版本，请在执行 `Docker run` 命令时设置 `NS_CPX_LITE` 环境变量。

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

以下示例基于 NetScaler CPX 映像创建了一个轻量级容器。

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

默认情况下，在较轻版本的 NetScaler CPX 上禁用使用 `newslog` 记录功能。要启用它，您必须在启动更轻版本的 NetScaler CPX 的同时将 `NS_ENABLE_NEWSLOG` 环境变量设置为 1。

以下示例说明如何在部署更轻版本的 NetScaler CPX 时启用使用 `newslog` 记录功能。

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

注意：CPX 的简化版仅支持单核 (`CPX_CORES=1`)。

使用 **Docker Compose** 部署 **NetScaler CPX** 实例

可以使用 Docker Compose 工具来预配一个 NetScaler CPX 实例或多个 NetScaler CPX 实例。要使用 Docker Compose 配置 NetScaler CPX 实例，必须先编写一个撰写文件。此文件指定了 NetScaler CPX 映像、您想要为 NetScaler CPX 实例打开的端口以及您的 NetScaler CPX 实例的权限。

重要提示

请确保您已在主机上安装了 Docker Compose 工具。

要置备多个 **NetScaler CPX** 实例，请执行以下操作：

1. 编写一个 Compose 文件，其中：

- **<service-name>** 是要置备的服务的名称。
- **image:<repository>:<tag>** 指示 NetScaler CPX 映像的存储库和版本。
- **privileged: true** 提供对 NetScaler CPX 实例的所有 root 权限。
- **cap_add** 提供对 NetScaler CPX 实例的网络权限。
- **<host_directory_path>** 指示 Docker 主机上要为 NetScaler CPX 实例装载的目录。
- **<number_processing_engine>** 是您希望 NetScaler CPX 实例启动的处理引擎数。对于每个附加处理引擎，请确保 Docker 主机包含相应数量的 vCPU 和内存量 (GB)。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。

编写文件的格式通常类似于：

```

1  <service-name>:
2  container_name:
3  image: <repository>:<tag>
4  ports:
5      - 22
6      - 9080
7      - 9443
8      - 161/udp
9      - 35021-35030
10 tty: true
11 cap_add:
12     - NET_ADMIN
13 ulimits:
14     core: -1
15 volumes:
16     - <host_directory_path>:/cpx
17 environment:
18     - EULA=yes
19     - CPX_CORES=<number_processing_engine>
20     - CPX_CONFIG='{
21 "YIELD":"Yes" }
22 '
23 <!--NeedCopy-->

```

```

1  CPX_0:
2  container_name: CPX_0
3  image: cpx:13.0-x.x
4  ports:
5      - 9443
6      - 22
7      - 9080

```

```
8     - 161/udp
9     - 35021-35030
10    tty: true
11    cap_add:
12      - NET_ADMIN
13    ulimits:
14      core: -1
15    volumes:
16      - /root/test:/cpx
17    environment:
18      - CPX_CORES=2
19      - EULA=yes
20 <!--NeedCopy-->
```

如果您想预置单个 NetScaler CPX 实例，则必须在撰写文件中添加以下行：`container_name:<name_of_container>`

运行以下命令来预置多个 NetScaler CPX 实例：

```
docker-compose -f <compose_file_name> scale <service-name>=<number of instances> up -d
docker-compose -f docker-compose.yml scale cpxlb=3 up -d
```

如果您想预置单个 NetScaler CPX 实例，请运行以下命令：`docker-compose -f <compose_file_name> up -d`

将 NetScaler CPX 实例添加到 NetScaler ADM

March 25, 2024

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

可以在第一次设置 ADM 时添加实例，也可在以后添加。

要添加实例，必须创建一个实例配置文件并指定每个实例的主机名或 IP 地址，或指定 IP 地址范围。此实例配置文件包含要添加到 Citrix ADM 的实例的用户名和密码。对于每个实例类型，都有一个默认的配置文件。例如，`ns-root-profile` 是 Citrix ADC 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。如果更改了实例的默认管理员凭据，可以为那些实例定义自定义实例配置文件。如果在发现实例后更改实例的凭据，则必须编辑实例配置文件或创建一个配置文件，然后重新发现实例。

必备条件

请务必执行以下操作：

- 已在 Citrix XenServer 上安装了 Citrix ADM 软件。有关详细信息，请参阅 [NetScaler ADM 文档](#)。

- 在 Docker 主机上安装了 NetScaler CPX 实例。

要将 **NetScaler CPX** 实例添加到 **ADM**，请执行以下操作：

- 在 Web 浏览器中，键入 **NetScaler Application Delivery Management** 的 IP 地址（例如 <http://192.168.100.1>）。
- 在 **User Name**（用户名）和 **Password**（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 **Networks**（网络） > **Instances**（实例） > **Citrix ADC**，然后单击 **CPX** 选项卡。
- 单击 **Add**（添加）以在 Citrix ADM 中添加新的 CPX 实例。
- 此时将打开“添加 **NetScaler CPX**”页。为以下参数输入值：
 - 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。
 - 选择要在其中部署实例的站点。
 - 选择代理。
 - 作为一种选择，您可以为实例输入键-值对。通过添加键-值对，您以后可以轻松搜索实例。

← Add Citrix ADC CPX

Enter Device IP Address Import from file

Enter one or more hostnames, IP addresses, and/or a range of IP addresses (for example, 10.102.40.30-10.102.40.45) using a comma separator.

Routable IP/ Docker IP*

 ?

Profile Name*

Site*

Agent

 >

Tags

<input type="text" value="Key"/>	<input type="text" value="Value"/>	+
----------------------------------	------------------------------------	---

- 单击确定。

注意

如果要重新发现实例，请导航到 **Networks** (网络) > **Instances** (实例) > **NetScaler ADC > CPX**，选择要重新发现的实例，然后从 **Select Action** (选择操作) 下拉列表中单击 **Rediscover** (重新发现)。

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

也可以使用环境变量将 NetScaler CPX 实例添加到 NetScaler ADM。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

下面是向 NetScaler ADM 添加 NetScaler CPX 实例的示例 `docker run` 命令。

```
1  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
2
3  <!--NeedCopy-->
```

配置 **NetScaler CPX**

November 23, 2023

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

使用命令行接口配置 **NetScaler CPX** 实例

访问 Docker 主机并登录实例的 SSH 提示窗口，如下图所示。用于登录 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窗口来运行 CLI 命令：**cli_script.sh** “<command>”

示例：

```

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

```

要从实例提示窗口注销，请键入 **logout**。

支持在 **NetScaler CPX** 中使用非默认密码

NetScaler CPX 支持为 root 用户帐户使用非默认密码，即 **nsroot**。部署 NetScaler CPX 后，系统会生成默认密码并将其分配给用户。SSH 用户的默认密码也会更新：**root** 和 **nsroot**。可以手动更改此默认密码。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保护系统的安全性。

重置密码后，新密码将用于 NITRO API 通信和 **cli_script.sh** 执行。

默认 root 用户帐户密码以纯文本形式存储在 NetScaler CPX 文件系统的 `/var/deviceinfo/random_id` 文件中。

使用以下语法通过凭据运行 **cli_script.sh**：

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

例如，要运行 **cli_script.sh** 以显示带有用户 **nsroot** 和密码 **Citrix123** 的 IP 地址，请使用以下命令：

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

使用 **NITRO API** 配置 **NetScaler CPX** 实例

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

要使用 **Nitro API** 来配置 **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** 来检索统计信息，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 的详细信息，请参阅 [REST Web 服务](#)。对于 NetScaler CPX，请使用 `CPX IP address :port`，其中提及 `netscaler-ip-address`。

使用作业配置 **NetScaler CPX** 实例

您可以通过在 NetScaler ADM 中创建和运行任务来配置 NetScaler CPX 实例。可以使用配置模板中的配置、提取其他设备上可用的配置以及使用文本文件中保存的配置。还可以记录通过使用其他实例的配置实用程序所做的配置。然后，NetScaler ADM 会显示相应的 CLI 命令供您使用。选择配置后，必须选择要在其上加载配置的 **NetScaler CPX** 实例，指定变量值并运行作业。

要使用作业配置 **NetScaler CPX** 实例，请执行以下操作：

1. 使用管理凭据登录到 NetScaler ADM。
2. 导航到 **Networks** (网络) > **Configuration Jobs** (配置作业)，然后单击 **Create Job** (创建作业)。
3. 指定所需值，并选择配置来源。还可以键入要运行的命令。

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

4. 选择要在其上运行配置的 NetScaler CPX 实例，然后单击“下一步”。

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

Add Instances Delete

<input type="checkbox"/>	IP Address	Name
<input type="checkbox"/>	172.17.0.150	10.102.31.190

Cancel Back Next Save and Exit

5. 指定执行设置并单击 Finish（完成）以对 NetScaler CPX 实例运行命令。如果要保存配置并稍后运行，请单击“保存并退出”。

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

November 23, 2023

您可以在 NetScaler CPX 实例上配置 AppFlow 功能，以收集网页性能数据、流量和用户会话级别信息以及应用程序性能监视和分析所需的数据库信息。这些数据记录发送至 NetScaler ADM，在那里可以查看您的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，您可以指定将流记录发送到的收集器。之后，您可以定义操作，这些操作是已配置的收集器的集合。然后，您可以配置一个或多个策略并将操作与每个策略关联。该策略告诉 NetScaler CPX 选择将流量记录发送到相关操作的请求。最后，您可以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。

可以进一步设置 AppFlow 参数以指定模板刷新时间间隔并启用 `httpURL`、`httpCookie` 和 `httpReferer` 信息的导出。在每个收集器上，必须指定 NetScaler CPX IP 地址作为导出器的地址。

配置实用程序提供了帮助用户定义策略和操作的工具。它确切地决定了 NetScaler CPX 如何将特定流量的记录导出到一组收集器（操作。）命令行界面为喜欢命令行的有经验的用户提供了一组相应的基于 CLI 的命令。

在监视记录之前，必须将 NetScaler CPX 实例添加到 NetScaler ADM。有关向 NetScaler ADM 添加 NetScaler CPX 实例的更多信息，请参阅[使用 NetScaler ADM 安装 NetScaler CPX 实例](#)。

启用 AppFlow

要使用 AppFlow 功能，必须首先将其启用。

要使用命令行界面启用 **AppFlow** 功能，请执行以下操作：

运行以下命令：

```
1 enable ns feature AppFlow
2 enable ns mode ulfd
```

指定收集器

收集器接收 Citrix ADC 生成的 AppFlow 记录。要发送 AppFlow 记录，必须至少指定一个收集器。默认情况下，收集器在 UDP 端口 4739 上侦听 IPFIX 消息。配置收集器时，可以更改默认端口。

要使用命令行界面指定收集器，请执行以下操作：

使用以下命令添加收集器：

```
1 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：

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

要使用命令行界面指定多个收集器，请执行以下操作：

使用以下命令添加相同的数据并将其发送到多个收集器：

```

1 add appflow collector <collector1> -IPAddress <IP> -Transport Logstream
2
3 add appflow collector <collector2> -IPAddress <IP> -Transport Logstream
4
5 add appflow action <action> -collectors <collector1> <collector2> -
  Transport Logstream
6
7 add appflow policy <policy> true <action> -Transport Logstream
8
9 bind lbvserver <lbvserver> -policy <policy> -priority <priority> -
  Transport Logstream

```

配置 **AppFlow** 操作

AppFlow 操作是一个集合收集器，如果关联的 AppFlow 策略匹配，则会将流记录发送到该收集器。

使用以下命令配置 AppFlow 操作：

```

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

```

要验证配置，请使用以下命令：

```

1 show appflow action

```

配置 **AppFlow** 策略

配置 AppFlow 操作后，必须随后配置 AppFlow 策略。AppFlow 策略基于由一个或多个表达式组成的规则。

要使用命令行界面配置 **AppFlow** 策略，请执行以下操作：

在命令提示符处，键入以下命令以添加 AppFlow 策略并验证配置：

```

1 add appflow policy <name> <rule> <action>
2
3 show appflow policy <name>

```

绑定 **AppFlow** 策略

要使策略生效，必须将其全局绑定，以使其适用于流经 NetScaler CPX 的所有流量。

要使用命令行界面全局绑定 **AppFlow** 策略，请执行以下操作：

使用以下命令全局绑定 AppFlow 策略：

```

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

```

使用以下命令验证配置：

```
1 show appflow global
```

使用配置文件配置 NetScaler CPX

November 23, 2023

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

可以在部署 NetScaler CPX 容器过程中提供一个静态配置文件作为输入文件。NetScaler CPX 容器启动过程中，将根据在静态配置文件中指定的配置对容器进行配置。此配置包括特定于 NetScaler ADC 的配置和 bash shell 命令，您可以在 NetScaler CPX 容器上动态运行这些命令。

静态配置文件的结构

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

静态配置文件是一个包括两个标记 `#NetScaler Commands` 和 `#Shell Commands` 的 `.conf` 文件。在 `#NetScaler Commands` 标签下，您必须添加所有 NetScaler ADC 命令才能在 NetScaler CPX 上配置 NetScaler ADC 的特定配置。在 `#Shell Commands` 标签下，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 ADC 命令和 shell 命令按照配置文件中指定的顺序在容器上运行。

重要：

- 可以在配置文件中多次重复标记。
- 标记不区分大小写。
- 配置文件必须存在于 `/etc` 目录中作为容器的文件系统中的 `cpx.conf` 文件。
- 配置文件还可以包括注释。必须在注释之前添加 `#` 字符。
- 如果使用配置文件部署 NetScaler CPX 容器时存在故障情形，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，配置文件会重新应用到容器上。

```
1 #NetScaler Commands
2
3 add lb vserver v1 http 1.1.1.1 80
4
5 add service s1 2.2.2.2 http 80
6
7 bind lb vserver v1 s1
8
9 #Shell Commands
```

```

10
11 touch /etc/a.txt
12
13 echo "this is a" > /etc/a.txt
14
15 #NetScaler Commands
16
17 add lb vserver v2 http
18
19 #Shell Commands
20
21 echo "this is a 1" >> /etc/a.txt
22
23 #NetScaler Commands
24
25 add lb vserver v3 http
26
27 #This is a test configuration file
28 <!--NeedCopy-->

```

要安装 NetScaler CPX 容器以及根据配置文件动态配置 NetScaler CPX 容器，请使用 `docker run` 命令中的 `-v` 选项装载静态配置文件：

```

1 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
2 <!--NeedCopy-->

```

NetScaler CPX 中的动态路由支持

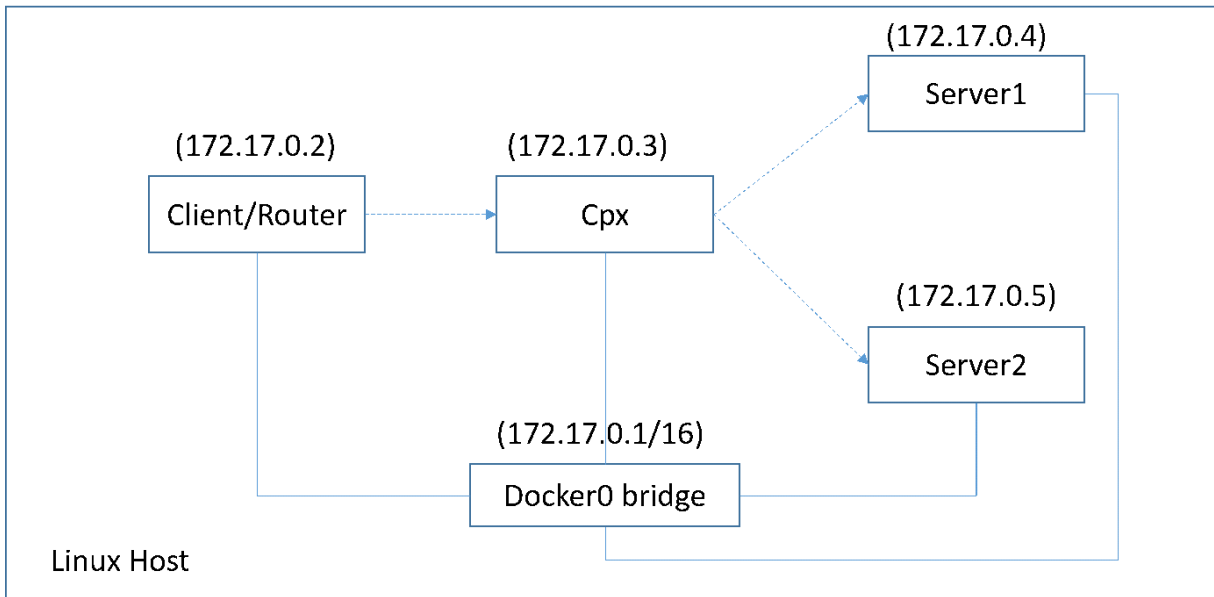
November 23, 2023

NetScaler CPX 支持 BGP 动态路由协议。动态路由协议的关键目标是根据绑定到虚拟服务器的服务的运行状况公告虚拟服务器的 IP 地址。它有助于上游路由器在通往按地形分布的虚拟服务器的多条路由中选择最佳路由。

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

在单主机网络中，客户端、服务器和 NetScaler CPX 实例都部署为同一 Docker 主机上的容器。所有容器都通过 `docker0` 桥接连接。在此环境中，NetScaler CPX 实例充当在同一 Docker 主机上置备为容器的应用程序的代理。有关 NetScaler CPX 主机联网模式部署的信息，请参阅[主机联网模式](#)。

下图说明了单主机拓扑。



在此拓扑中，使用 BGP 配置虚拟服务器并公告（基于服务的运行状况）到上游网络或路由器。

执行以下步骤，在具有桥接联网模式的单 Docker 主机上在 NetScaler CPX 上配置 BGP。

在 **NetScaler CPX** 上使用 **REST API** 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：

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

例如：

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

2. 使用以下命令登录容器：

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

3. 使用以下命令启用 BGP 功能：

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

4. 使用 `show ns ip` 命令获取 NSIP：

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

5. 使用以下命令添加虚拟服务器：

```
1 cli_script.sh "add lb vservice <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`:

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

从容器注销并将 BGP NITRO 命令从主机发送到端口 9080 上的 NSIP。

8. 配置 BGP 路由器:

例如, 如果要配置:

```
1 router bgp 100
2 Neighbour 172.17.0.2 remote-as 101
3 Redistribute kernel
```

按如下所示指定命令:

```
1 curl -u username:password http://<NSIP>:9080/nitro/v1/config/ -X
   POST --data 'object={
2   "routerDynamicRouting": {
3   "bgpRouter" : {
4   "localAS":100, "neighbor": [{
5     "address": "172.17.0.2", "remoteAS": 101 }
6   ], "afParams":{
7     "addressFamily": "ipv4", "redistribute": {
8     "protocol": "kernel" }
9   }
10  }
11  }
12  }
13  '
```

9. 使用以下 NITRO 命令将获取的 BGP 路由安装到 PE 中:

```
1 curl -u username:password http://<NSIP>:9080/nitro/v1/config/ --
   data 'object={
2   "params":{
3   "action":"apply" }
4   ,"routerDynamicRouting": {
5   "commandstring" : "ns route-install bgp" }
6   }
7   '
```

10. 使用以下 NITRO 命令验证 BGP 邻近状态:

```
1 curl -u username:password http://<NSIP>:9080/nitro/v1/config/
   routerDynamicRouting/bgpRouter
```

示例输出:

```
1 root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/
   nitro/v1/config/routerDynamicRouting/bgpRouter
2 {
```

```

3  "errorCode": 0, "message": "Done", "severity": "NONE", "
    routerDynamicRouting":{
4  "bgpRouter": [{
5  "localAS": 100, "routerId": "172.17.0.3", "afParams": [ {
6  "addressFamily": "ipv4" }
7  , {
8  "addressFamily": "ipv6" }
9  ], "neighbor": [ {
10 "address": "172.17.0.2", "remoteAS": 101, "ASOriginationInterval
    ": 15, "advertisementInterval": 30, "holdTimer": 90, "
    keepaliveTimer": 30, "state": "Connect", "singlehopBfd":
    false, "multihopBfd": false, "afParams": [ {
11 "addressFamily": "ipv4", "activate": true }
12 , {
13 "addressFamily": "ipv6", "activate": false }
14 ]

```

11. 使用以下命令验证通过 BGP 获取的路由是否安装在数据包引擎中：

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

12. 使用以下命令保存配置：

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

动态路由配置保存在 `/nsconfig/ZebOS.conf` 文件中。

为 NetScaler CPX 配置高可用性

November 23, 2023

具有任务关键型和业务关键型应用程序的系统必须持续可用，而不会出现单点故障。具有高可用性的系统可确保应用程序的持续可用性，而不会中断为用户提供的服务。NetScaler CPX 支持两个 NetScaler ADC 实例的高可用性部署，这可以保护服务免受计划外停机的影响，并确保出现故障时的业务连续性。配置高可用性后，您还可以升级 NetScaler CPX 软件，而不会对用户造成任何服务中断。

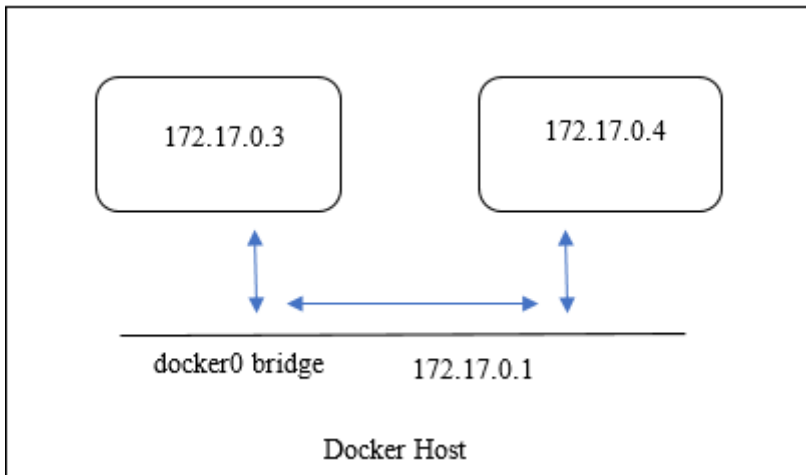
注意：

如果禁用内部用户帐户，则不支持 NetScaler CPX 功能的高可用性。

拓扑 1： 在具有桥接网络模式的单个 **Docker** 主机上部署 **NetScaler CPX** 实例

在此拓扑中，在具有桥接网络模式的同一 Docker 主机上创建了两个 NetScaler CPX 节点。两个节点都位于同一桥接网络上，节点可直接相互访问。

下图解释了此拓扑。



在此示例中，在同一 Docker 主机上创建了两个 NetScaler CPX 实例，即 CPX-1 (NSIP: 172.17.0.3) 和 CPX-2 (NSIP: 172.17.0.4)。要获得高可用性支持，您必须使用另一个节点的 NSIP 在两个 NetScaler CPX 实例上配置高可用性节点。

执行以下步骤，在桥接模式下的单个 docker 主机上的 NetScaler CPX 实例上配置高可用性支持。

1. 访问 Docker 主机并登录 NetScaler CPX 实例的 SSH 提示符。有关更多信息，请参阅[使用命令行界面配置 NetScaler CPX 实例](#)。
2. 使用以下命令在 CPX-1 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 172.17.0.4 [--inc enabled]'
```

3. 使用以下命令在 CPX-2 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 172.17.0.3 [--inc enabled]'
```

注意：

重新启动处于桥接网络模式的 NetScaler CPX 节点时，分配给 NetScaler CPX 的 IP 地址可能会根据主机上的 Docker 版本而变化。如果在重新启动 NetScaler CPX 后任一节点的 NSIP 发生变化，则即使保存了配置，现有的高可用性配置也将不起作用。在这种情况下，您必须再次在 NetScaler CPX 节点上配置高可用性。

拓扑 2：在具有桥接网络模式的不同 Docker 主机上部署 NetScaler CPX

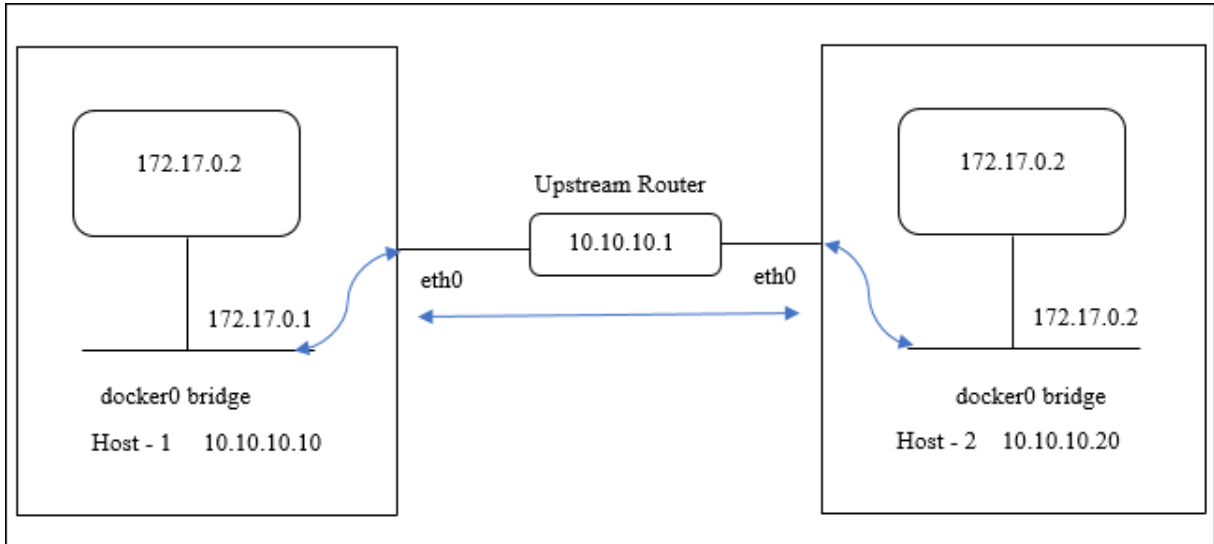
在此拓扑中，两个 NetScaler CPX 实例以桥接模式部署在两个可以相互访问的不同 Docker 主机上。在此部署中，NetScaler CPX 必须知道主机的 IP 地址。**HOST** 环境变量可以在配置 NetScaler CPX 时使用，让 NetScaler CPX 知道主机的 IP 地址。

您必须为 NetScaler CPX 节点设置端口映射。在创建 NetScaler CPX 节点时，您可以使用 **docker run** 命令的 **-p** 选项来启用所需端口的端口映射。

必须映射以下端口：

- UDP 3003
- TCP 3008
- TCP 8873

下图说明了在两个不同的 Docker 主机上以桥接模式部署两个 NetScaler CPX 实例的拓扑。



在此图中，蓝色直线表示两台主机之间的 CPX-HA 流量的流。

注意：在 Docker 主机上，只有一个 NetScaler CPX 可以形成高可用性对。同一主机上的任何其他 NetScaler CPX 都无法与另一台主机上的另一个 NetScaler CPX 形成高可用性对。

执行以下步骤以桥接模式在不同的 Docker 主机上部署 NetScaler ADC 实例，并使用示例拓扑配置高可用性支持。

在此示例中，host1 IP 地址配置为 10.10.10.10/24，

host2 IP 地址配置为 10.10.10.20/24。

1. 使用以下命令在 host1 上部署具有所需端口映射的 NetScaler CPX。

```
1 docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -p
  8873:8873 -p 3003:3003/udp -p 3008:3008 -e Host=10.10.10.10 cpx
  :latest
```

2. 使用带有主机 2 的 IP 地址的相同命令在 host2 上部署 NetScaler CPX。

```
1 docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -p
  8873:8873 -p 3003:3003/udp -p 3008:3008 -e HOST=10.10.10.20 cpx
  :latest
```

3. 使用以下命令在 CPX-1 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 10.10.10.20 -inc enabled'
```

4. 使用以下命令在 CPX-2 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 10.10.10.10 -inc enabled'
```

注意：在此部署中，必须使用高可用性节点的主机 IP 地址，而非高可用性节点的 NSIP 地址。

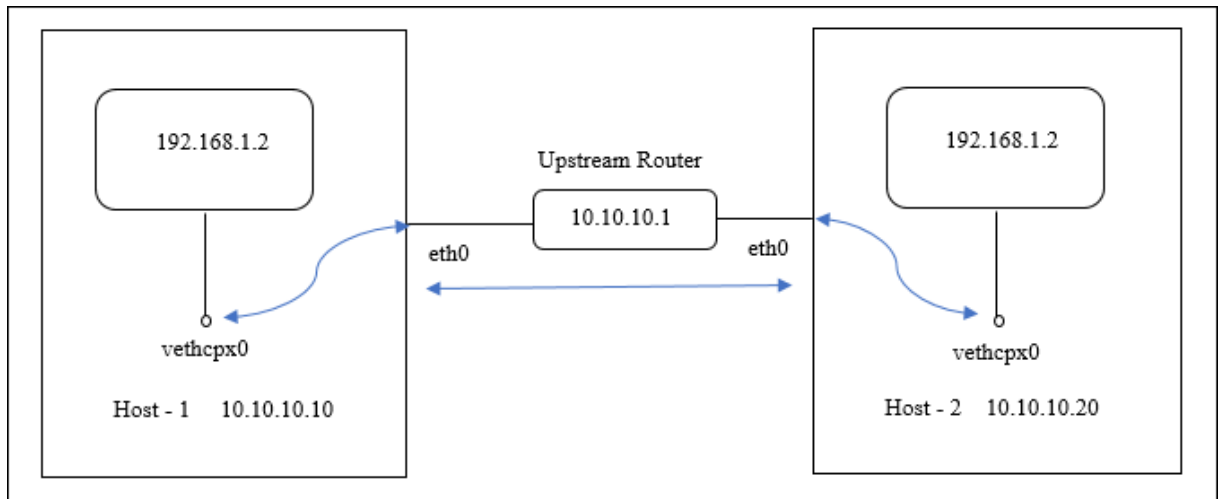
拓扑 3：在没有专用接口的主机网络模式下在不同的 **Docker** 主机上部署 **NetScaler CPX**

在此拓扑中，两个 NetScaler CPX 实例以主机模式部署在两个不同的 Docker 主机上，没有专用接口。主机必须可以相互访问。

在此部署中，NetScaler CPX 必须知道主机的 IP 地址。在配置 NetScaler CPX 期间，您可以使用 **HOST** 环境变量让它知道主机的 IP 地址。

您必须为 NetScaler CPX 节点设置端口映射。在创建 NetScaler CPX 节点时，您可以使用 **docker run** 命令的 **-p** 选项来启用所需端口的端口映射。

下图解释了拓扑结构。



在此图中，蓝色直线表示两台主机之间的 CPX-HA 流量的流。

注意：在 Docker 主机上，您只能部署一个主机模式 NetScaler CPX。

执行以下步骤部署 NetScaler CPX 实例并使用示例拓扑配置高可用性支持。

1. 使用所需的端口映射部署 NetScaler CPX，并使用以下命令在 host1 上部署。

```
1 docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 --
  net=host -e NS_NETMODE=HOST -e HOST=10.10.10.10 cpx:latest
```

2. 使用以下命令在主机 2 上部署 NetScaler CPX，IP 地址为 host2。

```
1 docker run -dt --privileged=true -e EULA=yes --ulimit core=-1
2 --net=host -e NS_NETMODE=HOST -e HOST=10.10.10.20 cpx:latest
```

3. 使用以下命令在 CPX-1 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 10.10.10.20 -inc enabled'
```

4. 使用以下命令在 CPX-2 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 10.10.10.10 -inc enabled'
```

拓扑 4：使用主机网络模式和专用接口在不同的 **Docker** 主机上部署 **CPX**

在此拓扑中，两个 NetScaler CPX 实例以主机联网模式部署在不同的 Docker 主机上。主机必须有多个接口。您可以使用 **CPX_NW_DEV** 环境变量为 NetScaler CPX 指定专用接口。

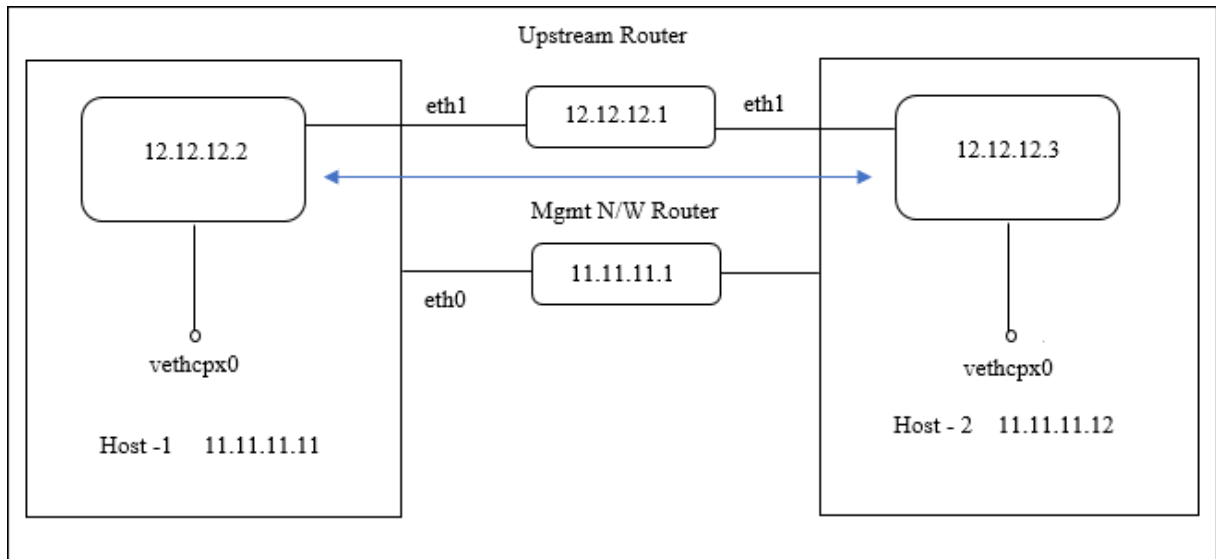
有关使用 CPX_NW_DEV 环境变量为 NetScaler CPX 分配专用网络接口的更多信息，请参阅[使用 docker run 命令部署 NetScaler CPX 实例](#)。

部署在不同 Docker 主机上的 NetScaler CPX 必须能够通过专用接口在此数据网络上相互访问。

此配置允许高可用性节点通过直接在端口 3003、3008 和 8873 上进行通信来交换检测信号消息和同步配置文件。主机上不需要 NAT 规则。在主机模式下创建的 NetScaler CPX 的默认 NSIP 在两个节点上是相同的。因此，还必须指定 **NS_IP** 和 **NS_GATEWAY** 信息。

在此示例中，在两台不同的主机上创建了两个主机模式 NetScaler CPX。NetScaler CPX 实例在两台主机上拥有 **eth1** 接口，并且 **eth1** 接口连接到同一个网络。

下图解释了拓扑结构。在此图中，蓝色箭头表示连接到 **eth1** 接口的网络上的 CPX-HA 流量的流。



注意：在 Docker 主机上，您只能部署一个主机模式 NetScaler CPX。

执行以下步骤部署 NetScaler CPX 实例并使用示例拓扑配置高可用性支持。

1. 使用以下命令在 host1 上以主机模式部署 NetScaler CPX。

```
1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -
  e CPX_NW_DEV=eth1 -e NS_IP='12.12.12.2' -e NS_GATEWAY='
  12.12.12.9' -e EULA=yes --ulimit core=-1 cpx:latest
```

2. 使用以下命令在主机模式下在 host2 上部署 NetScaler CPX。

```
1 docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -
  e CPX_NW_DEV=eth1 -e NS_IP='12.12.12.3' -e NS_GATEWAY='
  12.12.12.10' -e EULA=yes --ulimit core=-1 cpx:latest
```

注意：必须为两个 NetScaler CPX 节点配置静态路由，才能到达另一个 NetScaler CPX 节点以交换心跳消息和同步配置文件。

3. 使用以下命令在 CPX-1 实例上配置高可用性节点。

```
1 cli_script.sh 'add ha node 1 12.12.12.3 [-inc enabled]'
```

4. 使用以下命令在 CPX-2 实例上配置高可用性节点。

```
1 cli_script.sh 'add high availability node 1 12.12.12.2 [-inc
  enabled]'
```

配置 Docker 日志记录驱动程序

November 23, 2023

Docker 包括名为“日志记录驱动程序”的日志记录机制以帮助您从正在运行的容器中获取信息。可以将 NetScaler CPX 容器配置为将其生成的日志转发到 Docker 日志记录驱动程序。有关 Docker 日志记录驱动程序的详细信息，请参阅 <https://docs.docker.com/config/containers/logging/configure/>。

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。使用 `docker run` 命令启动 NetScaler CPX 容器时，可以将其配置为使用 `--log-driver` 选项将生成的所有日志转发到 Docker 日志记录驱动程序。如果日志记录驱动程序具有可配置的参数，则可以使用 `--log-opt <NAME>=<VALUE>` 选项对其进行设置。

在下例中，NetScaler CPX 容器配置为使用 `syslog` 作为日志记录驱动程序转发生成的所有日志。

```
1 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
2 <!--NeedCopy-->
```

同样，在下例中，NetScaler CPX 容器配置为使用 `Splunk` 作为日志记录驱动程序转发生成的所有日志。

```

1 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
2 <!--NeedCopy-->

```

升级 NetScaler CPX 实例

November 23, 2023

可以通过关闭 NetScaler 实例、在同一装载点上安装最新版本的 NetScaler，然后删除旧实例来升级该实例。装载点是在主机上装载 **/cpx** 目录的目录。

例如，要在主机的 **/var/cpx** 目录中装载现有 NetScaler CPX 实例的 **/cpx** 目录，装载点为 **/var/cpx**，NetScaler CPX 装载目录为 **/cpx**，如下所示：

```

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

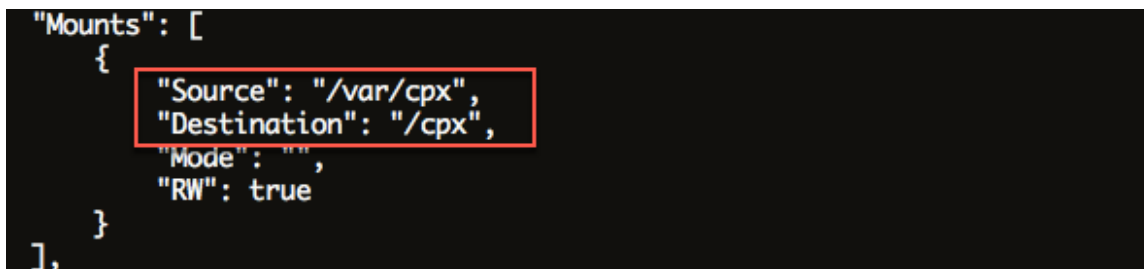
```

必备条件

请确保您具有：

- 具有您装载了现有 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）显示有关主机目录的信息。

该命令的输出提供有关容器配置的详细信息（包括卷）。在“**Mounts**”条目里，“**Source**”子条目显示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 映像文件并加载 NetScaler CPX Docker 映像。要加载映像，请导航到保存 Docker 映像文件的目录。使用 `docker load -i <image_name>` 命令加载映像。在加载 NetScaler CPX 映像后，可以输入 `docker images` 命令以显示与映像有关的信息：


```

1 root@ubuntu:~# docker load -i cpx-13.0-x.x.gz
2
3 root@ubuntu:~# docker images
4
5 REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE
6
7 cpx 13.0-x.x 2e97aadf918b 43 hours ago 414.5 MB
8 <!--NeedCopy-->

```

升级 NetScaler CPX 实例

1. 停止现有 NetScaler CPX 实例，方法是输入 `docker stop <containerName>` 命令，其中 `<containerName>` 为 NetScaler CPX 实例的名称。

```

1 root@ubuntu:~# docker stop mycpx
2 mycpx
3 <!--NeedCopy-->

```

2. 使用 `docker run` 命令，从加载到主机上的 NetScaler CPX 映像部署最新的 NetScaler CPX 实例。请确保在用于现有 NetScaler CPX 实例的同一装载点（例如 `/var/cpx:/cpx`）上部署该实例。

```

1 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
2 <!--NeedCopy-->

```

可以输入 `docker ps` 命令以便验证部署的 NetScaler CPX 实例是最新版本。

```

1 ````
2 root@ubuntu:~# docker ps
3
4 CONTAINER ID IMAGE COMMAND PORTS
5 CREATED STATUS NAMES
6 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
7 <!--NeedCopy--> ````

```

3. 验证您已部署了正确的 NetScaler CPX 实例后，输入 `docker rm <containerName>` 命令删除较旧实例。

```

1 root@ubuntu:~# docker rm mycpx
2 mycpx
3 <!--NeedCopy-->

```

在 NetScaler CPX 实例中使用通配符虚拟服务器

November 23, 2023

当您配置 NetScaler ADC 实例时，Docker 引擎仅向 NetScaler CPX 实例分配一个私有 IP 地址（单个 IP 地址）。NetScaler ADC 实例的三个 IP 功能会多路复用到一个 IP 地址。此单一 IP 地址使用不同的端口号来执行 NSIP、SNIP 和 VIP 功能。

Docker 引擎分配的单一 IP 地址是动态的。请使用单一 IP 地址或使用 127.0.0.1 IP 地址来添加负载均衡 (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，创建通配符虚拟服务器时，NetScaler CPX 将替换通配符虚拟服务器的已分配 IP 地址。分配的 IP 地址为 127.0.0.1，它被 Docker 引擎分配给 NetScaler CPX 实例的 NSIP 所取代。

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在辅助 NetScaler CPX 实例上配置通配符虚拟服务器。它无需在 Docker 引擎分配给 NetScaler CPX 实例的 NSIP 上配置虚拟服务器。

注意事项：

- 请确保您分配给通配符虚拟服务器的端口号没有被部署中的任何其他虚拟服务器使用。
- 如果内部服务已在使用分配给通配符虚拟服务器的端口号，添加通配符虚拟服务器将失败。
- 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，请在命令提示窗口中输入以下命令：

```
1 add lb vserver <name> <serviceType> 127.0.0.1 <port>
2
3 add lb vserver testlbvserver HTTP 127.0.0.1 30000
4 <!--NeedCopy-->
```

要创建通配符内容交换虚拟服务器，请在命令提示窗口中输入以下命令：

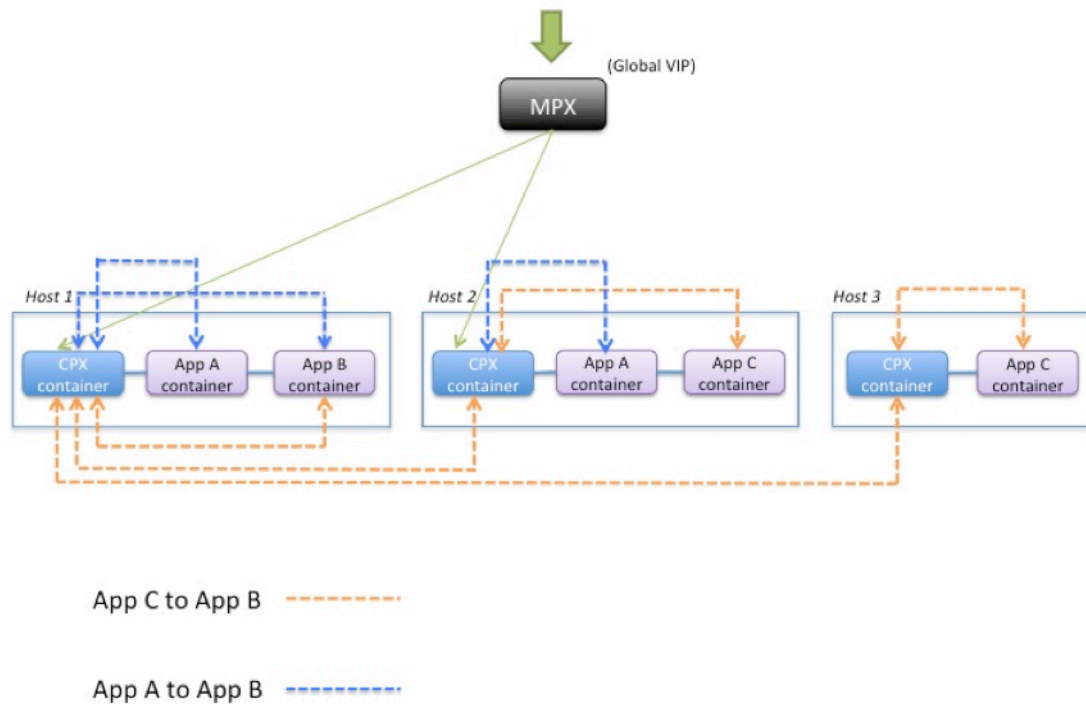
```
1 add cs vserver <name> <serviceType> 127.0.0.1 <port>
2
3 add cs vserver testcsvserver HTTP 127.0.0.1 30000
4 <!--NeedCopy-->
```

将 NetScaler CPX 部署为代理以支持东西通信流

November 23, 2023

在此部署中，NetScaler CPX 实例充当代理以便能在位于多个主机上的应用程序容器之间进行通信。NetScaler CPX 实例与多个主机上的应用程序一起置备，为通信提供最短路径。

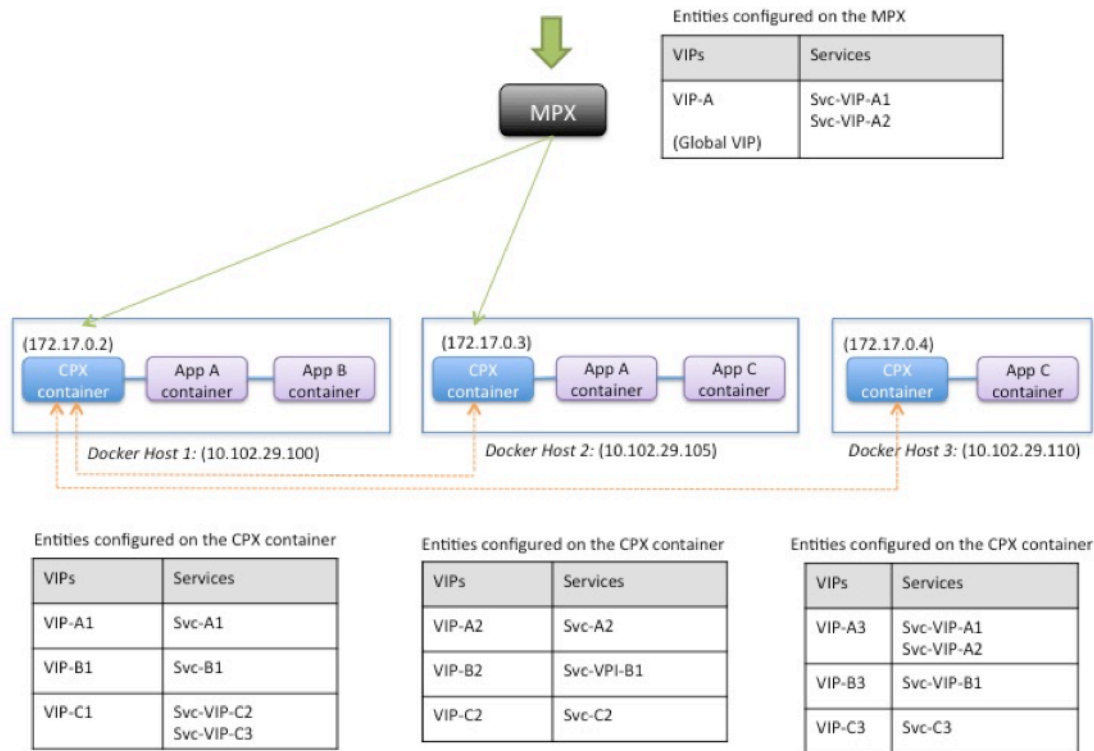
下图说明了两个应用程序之间通过 NetScaler CPX 实例进行的通信流。



此图显示了应用程序 C 与应用程序 B 之间以及应用程序 A 与应用程序 B 之间的通信流。当应用程序 C（位于任何主机中）向 B 发送请求时，与应用程序 C 在同一主机上的 NetScaler CPX 容器先接收该请求。然后 NetScaler CPX 容器将流量传送到与应用程序 B 在同一主机上托管的 NetScaler CPX 容器，之后该流量被转发至应用程序 B。应用程序 A 向应用程序 B 发送请求时，使用的是类似的流量路径。

在此示例中，NetScaler MPX 还部署为允许通过全局 VIP 将来自 Internet 的流量传送到应用程序。来自 NetScaler MPX 的流量在 NetScaler CPX 容器上接收，然后该容器在应用程序容器之间分发流量。

下图说明了此拓扑，此处需要设置配置才能进行通信。



下表列出了此配置示例中对 NetScaler CPX 实例配置的 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				

为了配置此方案示例，请在所有三个 Docker 主机上，创建 NetScaler CPX 容器时在 Linux shell 提示窗口中运行以下命令：

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

使用 NetScaler ADM 中的作业功能或使用 NITRO API 来运行以下命令。

对于 Docker 主机 1 上的 NetScaler CPX 实例：

```
1 add lb vserver VIP-A1 HTTP 172.17.0.2 30000
2 add service svc-A1 10.102.29.100 HTTP 80
3 bind lb vserver VIP-A1 svc-A1
4 add lb vserver VIP-B1 HTTP 172.17.0.2 30001
5 add service svc-B1 10.102.29.100 HTTP 90
6 bind lb vserver VIP-B1 svc-B1
7 add lb vserver VIP-C1 HTTP 172.17.0.2 30002
8 add service svc-VIP-C2 10.102.29.105 HTTP 30002
9 add service svc-VIP-C3 10.102.29.110 HTTP 30002
10 bind lb vserver VIP-C1 svc-VIP-C2
11 bind lb vserver VIP-C1 svc-VIP-C3
12 <!--NeedCopy-->
```

对于 Docker 主机 2 上的 NetScaler CPX 实例：

```
1 add lb vserver VIP-A2 HTTP 172.17.0.3 30000
2 add service svc-A2 10.102.29.105 HTTP 80
3 bind lb vserver VIP-A2 svc-A2
4 add lb vserver VIP-B2 HTTP 172.17.0.3 30001
5 add service svc-VIP-B1 10.102.29.100 HTTP 30001
6 bind lb vserver VIP-B2 svc-VIP-B1
7 add lb vserver VIP-C2 HTTP 172.17.0.3 30002
8 add service svc-C2 10.102.29.105 HTTP 70
9 bind lb vserver VIP-C2 svc-C2
10 <!--NeedCopy-->
```

对于 Docker 主机 3 上的 NetScaler CPX 实例：

```
1 add lb vserver VIP-A3 HTTP 172.17.0.4 30000
2 add service svc-VIP-A1 10.102.29.100 HTTP 30000
3 add service svc-VIP-A2 10.102.29.105 HTTP 30000
4 bind lb vserver VIP-A3 svc-VIP-A1
5 bind lb vserver VIP-A3 svc-VIP-A2
6 add lb vserver VIP-B3 HTTP 172.17.0.4 30001
7 add service svc-VIP-B1 10.102.29.100 HTTP 30001
8 bind lb vserver VIP-B3 svc-VIP-B1
9 add lb vserver VIP-C3 HTTP 172.17.0.4 30002
10 add service svc-C3 10.102.29.110 HTTP 70
11 bind lb vserver VIP-C3 svc-C3
12 <!--NeedCopy-->
```

在单主机网络中部署 NetScaler CPX

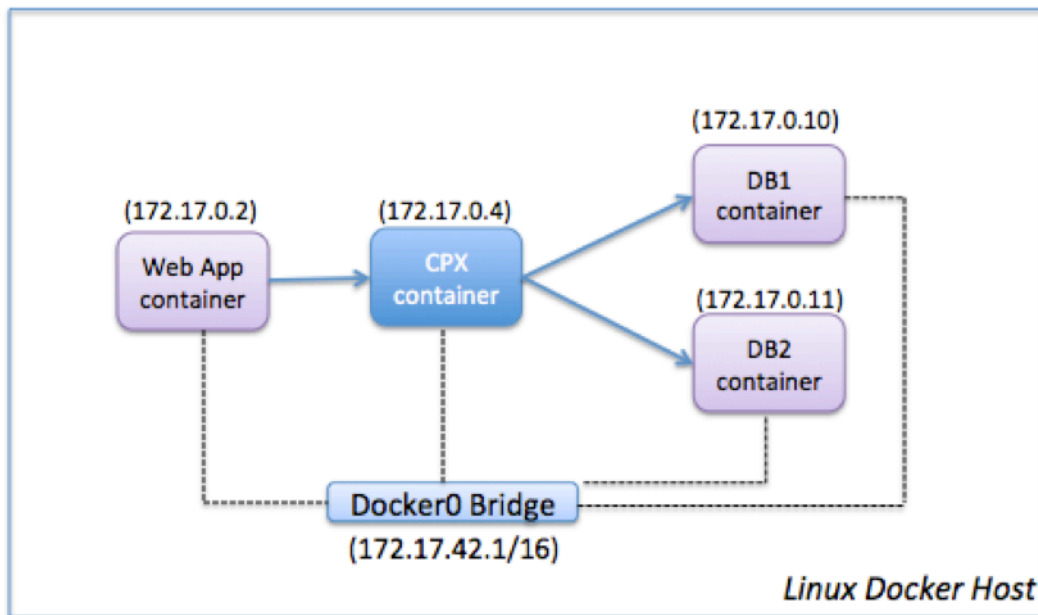
November 23, 2023

在单主机网络中，NetScaler CPX 实例充当同一主机上的应用程序容器之间的代理。在此容量中，NetScaler CPX 实例为基于容器的应用程序提供可扩展性和安全性。此外，它还优化性能，并提供有关遥测数据的洞察信息。

在单主机网络中，客户端、服务器和 NetScaler CPX 实例都部署为同一 Docker 主机上的容器。所有容器都通过 docker0 桥接连接。

在此环境中，NetScaler CPX 实例充当在同一 Docker 主机上置备为容器的应用程序的代理。

下图说明了单主机拓扑。



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。

为了让 Web 应用程序能够通过 NetScaler CPX 与数据库容器通信，必须先在 NetScaler CPX 容器上配置两个服务以代表两个服务器。然后使用 NetScaler CPX IP 地址和非标准 HTTP 端口（例如 81）（NetScaler CPX 保留标准 HTTP 端口 80 用于 NITRO 通信）来配置虚拟服务器。

在此拓扑中，不必配置任何 NAT 规则，因为客户端和服务器在同一网络中。

要配置此方案，请使用 Citrix ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```

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

```

5   bind lb vserver cpx-vip db2
6   <!--NeedCopy-->

```

在多主机网络中部署 NetScaler CPX

November 23, 2023

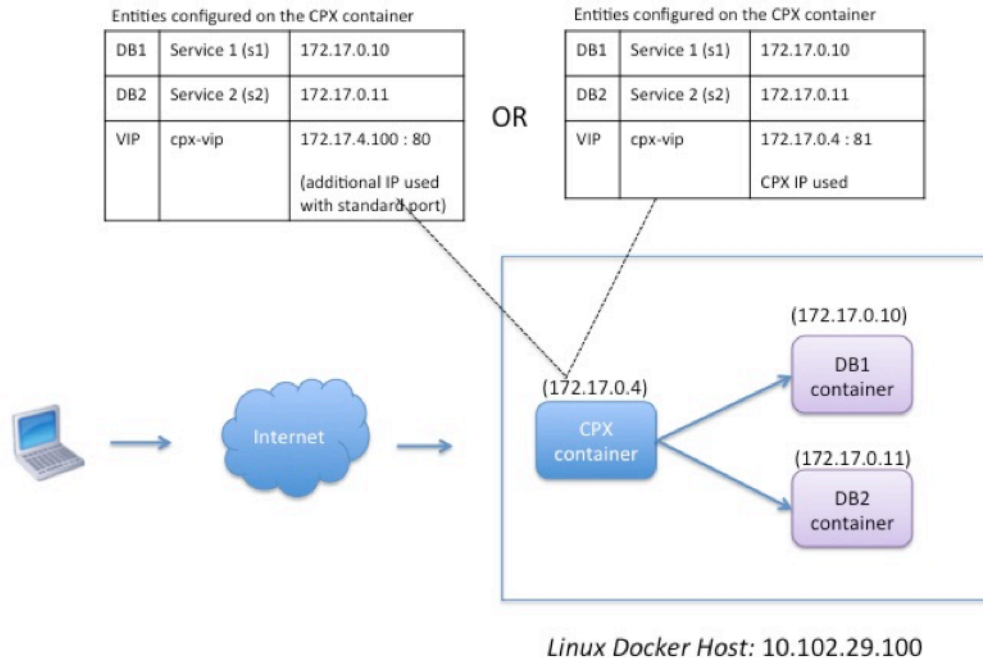
可以在数据中心的生产部署中配置多主机网络中的 NetScaler CPX 实例，在那里它提供负载平衡功能。此外它还提供监视功能和分析数据。

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。可以在生产部署中使用多主机拓扑，在那里 NetScaler CPX 实例为一组基于容器的应用程序和服务器甚至是物理服务器执行负载平衡。

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

在此拓扑中，在同一 Docker 主机上置备 NetScaler CPX 实例和数据库服务器，但客户端流量源自网络的其他地方。此拓扑可用于生产部署中，在那里 NetScaler CPX 实例为一组基于容器的应用程序或服务器执行负载平衡。

下图说明了此拓扑。



在此示例中，在 IP 地址为 10.102.29.100 的同一 Docker 主机上置备 NetScaler CPX 实例 (172.17.0.4) 与两台服务器 DB1 (172.17.0.10) 和 DB2 (172.17.0.11)。客户端位于网络的其他地方。

源自 Internet 的客户端请求通过在 NetScaler CPX 实例上配置的 VIP 接收，之后该实例在两台服务器之间分发请求。

有两种配置此拓扑的方法：

方法 1： 将其他 IP 地址和标准端口用于 VIP

1. 使用额外的 IP 地址在 NetScaler CPX 容器上配置 VIP。
2. 为 Docker 主机配置其他 IP 地址。
3. 配置 NAT 规则以将 Docker 主机的其他 IP 地址上接收的所有流量转发至 VIP 的其他 IP 地址。
4. 在 NetScaler CPX 实例上，将两台服务器配置为服务。
5. 最后将服务绑定至 VIP。

请注意，此配置示例中，10.x.x.x 网络表示公用网络。

要配置此方案示例，请使用 Citrix ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```
1 add service s1 172.17.0.10 HTTP 80
2 add service s2 172.17.0.11 HTTP 80
3 add lb vserver cpx-vip HTTP 172.17.4.100 80
4 bind lb vserver cpx-vip s1
5 bind lb vserver cpx-vip s2
6 <!--NeedCopy-->
```

通过在 Linux shell 提示窗口运行以下命令来为 Docker 主机配置其他公用 IP 地址并配置 NAT 规则。

```
1 ip addr add 10.102.29.103/24 dev eth0
2 iptables -t nat -A PREROUTING -p ip -d 10.102.29.103 -j DNAT --to-
  destination 172.17.4.100
3 <!--NeedCopy-->
```

方法 2： 将 NetScaler CPX IP 地址用于 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 用于 VIP。
2. 将服务绑定至 VIP。
3. 配置 NAT 规则以将 Docker 主机的端口 50000 上接收的所有流量转发至 VIP 和端口 81。

为了配置此方案示例，请在所有三个 Docker 主机上，创建 NetScaler CPX 容器时在 Linux shell 提示窗口中运行以下命令：

```
1 docker run -dt -p 22 -p 80 -p 161/udp -p 50000:81 --ulimit core=-1
  --privileged=true cpx:6.2
2
3 <!--NeedCopy-->
```

配置 NetScaler CPX 实例后，使用 NetScaler ADM 中的 Jobs 功能或使用 NITRO API 运行以下命令：


```

1   add service s1 172.17.0.10 http 80
2   add service s2 172.17.0.11 http 80
3   add lb vserver cpx-vip HTTP 172.17.0.4 81
4   bind lb vserver cpx-vip s1
5   bind lb vserver cpx-vip s2
6   <!--NeedCopy-->

```

注意：

如果在置备 NetScaler CPX 实例过程中未配置端口映射，请通过在 Linux shell 提示窗口中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

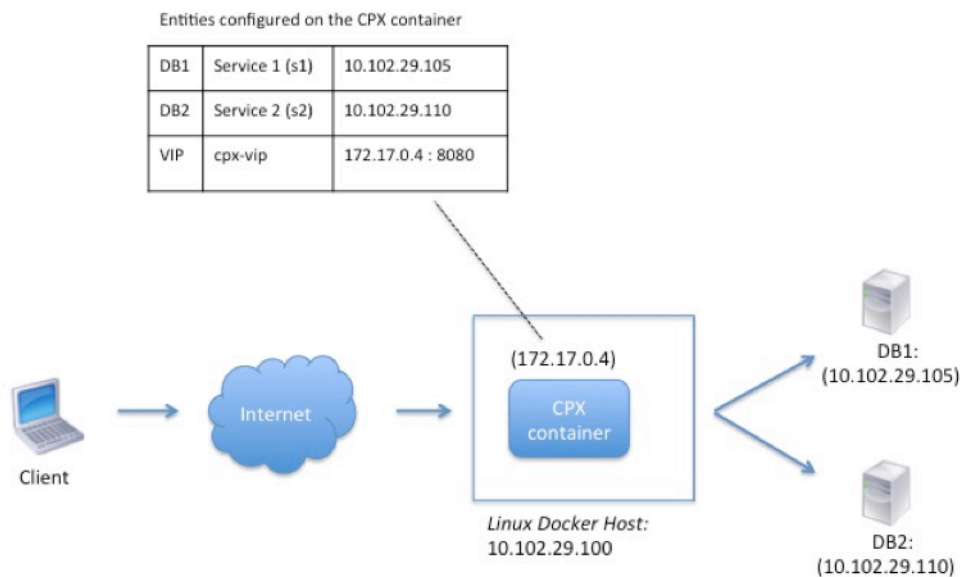
```

拓扑 2：具有物理服务器和客户端的 NetScaler CPX

在此拓扑中，在 Docker 主机上只置备了 NetScaler CPX 实例。客户端和服务不是基于容器的，且位于网络的其他地方。

在此环境中，您可以配置 NetScaler CPX 实例以在物理服务器之间对流量执行负载平衡。

下图说明了此拓扑。



在此示例中，NetScaler CPX 容器 (172.17.0.4) 位于客户端和物理服务器之间，充当代理。服务器 DB1 (10.102.29.105) 和 DB2 (10.102.29.110) 位于网络上 Docker 主机以外的地方。客户端请求源自 Internet 并在 NetScaler CPX 上接收，该 CPX 在两台服务器之间分发请求。

为了能够在客户端和服务器之间通过 NetScaler CPX 进行通信，必须先在创建 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代表两台服务器。最后，使用 NetScaler CPX IP 地址和非标准的映射 HTTP 端口 8080 来配置虚拟服务器。

请注意，配置示例中，10.x.x.x 网络表示公用网络。

为了配置此方案示例，请在创建 NetScaler CPX 容器时在 Linux shell 提示窗口中运行以下命令：

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

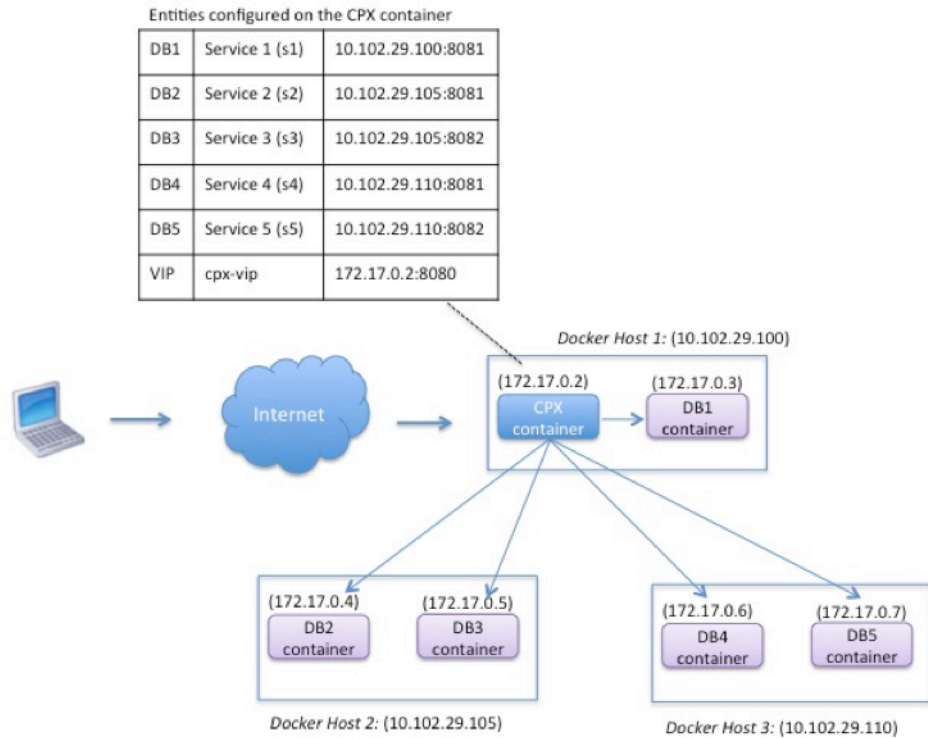
然后，使用 NetScaler ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```
1     add service s1 HTTP 10.102.29.105 80
2     add service s2 HTTP 10.102.29.110 80
3     add lb vserver cpx-vip HTTP 172.17.0.4 8080
4     bind lb vserver cpx-vip s1
5     bind lb vserver cpx-vip s2
6 <!--NeedCopy-->
```

拓扑 3：在不同的主机上置备 NetScaler CPX 和服务

在此拓扑中，在不同的 Docker 主机上置备 NetScaler CPX 实例和数据库服务器，客户端流量源自 Internet。此拓扑可用于生产部署中，在那里 NetScaler CPX 实例为一组基于容器的应用程序或服务器执行负载平衡。

下图说明了此拓扑。



在此示例中，在 IP 地址为 10.102.29.100 的同一 Docker 主机上置备了 NetScaler CPX 实例和服务 (DB1)。在两个不同的 Docker 主机 10.102.29.105 和 10.102.29.110 上置备了四台其他的服务器 (DB2、DB3、DB4 和 DB5)。

源自 Internet 的客户端请求在 NetScaler CPX 实例上接收，之后该实例在五台服务器之间分发请求。为了实现此通信，必须进行以下配置：

1. 创建您的 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 转发至主机上的端口 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，将五台服务器配置为服务。必须使用各个 Docker 主机 IP 地址和映射的端口组合来设置这些服务。
3. 在 NetScaler CPX 实例上配置 VIP 以接收客户端请求。此 VIP 由映射到主机的端口 8080 的 NetScaler CPX IP 地址和端口 8080 来表示。
4. 最后将服务绑定至 VIP。

请注意，配置示例中，10.x.x.x 网络表示公用网络。

为了配置此方案示例，请在创建 NetScaler CPX 容器时在 Linux shell 提示窗口中运行以下命令：

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

使用 NetScaler ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```

1   add service s1 10.102.29.100 HTTP 8081
2   add service s2 10.102.29.105 HTTP 8081
3   add service s3 10.102.29.105 HTTP 8082
4   add service s4 10.102.29.110 HTTP 8081
5   add service s5 10.102.29.110 HTTP 8082
6   add lb vserver cpx-vip HTTP 172.17.0.2 8080
7   bind lb vserver cpx-vip s1
8   bind lb vserver cpx-vip s2
9   bind lb vserver cpx-vip s3
10  bind lb vserver cpx-vip s4
11  bind lb vserver cpx-vip s5
12  <!--NeedCopy-->

```

为 NetScaler CPX 部署对网络的直接访问

November 23, 2023

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

为了实现此通信，必须先在 docker0 桥接上配置公用 IP 地址。然后，从网络端口 eth0 删除公用 IP 地址，并将该网络端口绑定到 docker0 桥接。

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

在示例配置中，10.x.x.x 网络表示公用网络。

要配置此方案，请在 Linux shell 提示窗口中运行以下命令：

```

1   ip addr add 10.102.29.100/24 dev docker0;
2   ip addr del 10.102.29.100/24 dev eth0;
3   brctl addif docker0 eth0;
4   ip route del default;
5   ip route add default via 10.102.29.1 dev docker0
6   <!--NeedCopy-->

```

使用 Citrix ADM 中的作业功能或使用 NITRO API 运行以下命令：

```

1   add service s1 172.17.0.8 http 80
2   add service s2 172.17.0.9 http 80
3   add lb vserver cpx-vip HTTP 10.102.29.102 80
4   bind lb vserver cpx-vip s1
5   bind lb vserver cpx-vip s2
6   <!--NeedCopy-->

```

使用 ConfigMap 在 Kubernetes 中配置 NetScaler CPX

November 23, 2023

在 Kubernetes 中，可以使用 ConfigMap 配置 NetScaler CPX 实例。使用 ConfigMap，可以在实例启动过程中动态配置 NetScaler CPX 实例。

创建一个 `cpx.conf` 配置文件，其中包含您要在 NetScaler CPX 实例上动态运行的 NetScaler ADC 特定的配置和 `bash shell` 命令。配置文件结构需要两种类型的标记，即 `#Citrix ADC Commands` 和 `#Shell Commands`。在 `#Citrix ADC Commands` 标签下，您必须添加所有 NetScaler ADC 命令才能在 NetScaler CPX 实例上配置 NetScaler ADC 的特定配置。在标记 `#Shell Commands` 下，必须添加要在 NetScaler CPX 实例上运行的 `shell` 命令。

重要：

- 可以在配置文件中多次重复标记。
- 配置文件还可以包括注释。请在注释之前添加 `#` 字符。
- 标记不区分大小写。
- 如果使用配置文件部署 NetScaler CPX 容器时存在故障情形，故障将记录在 `ns.log` 文件中。
- NetScaler CPX 实例启动后，如果更改了 ConfigMap，则仅当重新启动 NetScaler CPX 实例时才能应用更新后的配置。

下面是示例配置文件：

```
1 #Citrix ADC Commands
2 add lb vserver v1 http 1.1.1.1 80
3 add service s1 2.2.2.2 http 80
4 bind lb vserver v1 s1
5 #Shell Commands
6 touch /etc/a.txt
7 echo "this is a" > /etc/a.txt
8 #Citrix ADC Commands
9 add lb vserver v2 http
10 #Shell Commands
11 echo "this is a 1" >> /etc/a.txt
12 #Citrix ADC Commands
13 add lb vserver v3 http
14 <!--NeedCopy-->
```

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

```
1 kubectl create configmap cpx-config --from-file=cpx.conf
2 <!--NeedCopy-->
```

在上例中，可以根据配置文件 `cpx.conf` 创建 ConfigMap `cpx-config`。然后可以在使用的 YAML 文件中使用此 ConfigMap 来部署 NetScaler CPX 实例。

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:

```

1  apiVersion: v1
2  data:
3    cpx.conf: |
4      #Citrix ADC Commands
5        add lb vserver v1 http 1.1.1.1 80
6        add service s1 2.2.2.2 http 80
7        bind lb vserver v1 s1
8      #Shell Commands
9        touch /etc/a.txt
10       echo "this is a" > /etc/a.txt
11       echo "this is the file" >> /etc/a.txt
12       ls >> /etc/a.txt
13     #Citrix ADC Commands
14       add lb vserver v2 http
15     #Shell Commands
16       echo "this is a 1" >> /etc/a.txt
17     #Citrix ADC Commands
18       add lb vserver v3 http
19     #end of file
20  kind: ConfigMap
21  metadata:
22    creationTimestamp: 2017-12-26T06:26:50Z
23    name: cpx-config
24    namespace: default
25    resourceVersion: "8865149"
26    selfLink: /api/v1/namespaces/default/configmaps/cpx-config
27    uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
28  <!--NeedCopy-->
```

可以在使用的 YAML 文件中指定创建的 ConfigMap `cpx-config` 来部署 NetScaler CPX 实例，如下所示：

```

1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: cpx-1
5    labels:
6      app: cpx-daemon
7    annotations:
8      NETSCALER_AS_APP: "True"
9  spec:
10   hostNetwork: true
11   containers:
12   - name: cpx
13     image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-36.28"
14     securityContext:
15       privileged: true
16     volumeMounts:
17     - name: config-volume
```

```

18     mountPath: /cpx/conf
19     env:
20     - name: "EULA"
21       value: "yes"
22     - name: "NS_NETMODE"
23       value: "HOST"
24     - name: "kubernetes_url"
25       value: "https://10.90.248.101:6443"
26     - name: "NS_MGMT_SERVER"
27       value: "10.90.248.99"
28     - name: "NS_MGMT_FINGER_PRINT"
29       value: "19:71:A3:36:85:0A:2B:62:24:65:0F:7E:72:CC:DC:AD:B8:BF
30         :53:1E"
31     - name: "NS_ROUTABLE"
32       value: "FALSE"
33     - name: "KUBERNETES_TASK_ID"
34       valueFrom:
35         fieldRef:
36           fieldPath: metadata.name
37     imagePullPolicy: Never
38     volumes:
39     - name: config-volume
40       configMap:
41         name: cpx-config
42 <!--NeedCopy-->

```

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

部署 NetScaler CPX 作为 Kubernetes 节点的本地 DNS 缓存

November 23, 2023

Kubernetes 群集中的应用程序 pod 依靠 DNS 与其他应用程序 pod 进行通信。来自 Kubernetes 群集内的应用程序的 DNS 请求由 Kubernetes DNS (kube-dns) 进行处理。由于微服务体系结构的广泛采用，Kubernetes 群集内的 DNS 请求率不断增加。因此，Kubernetes DNS (kube-dns) 会负担过重。现在，您可以在每个 Kubernetes 节点上部署 NetScaler CPX 作为本地 DNS 缓存，并将来自节点中应用程序容器的 DNS 请求转发到 NetScaler CPX。因此，您可以更快地解析 DNS 请求，并显著降低 Kubernetes DNS 的负载。

要部署 NetScaler CPX，使用 Kubernetes DaemonSet 实体在 Kubernetes 群集中的每个节点上调度 NetScaler CPX 容器。Kubernetes DaemonSet 确保群集中的每个 Kubernetes 节点上都有一个 NetScaler CPX 实例。

要让应用程序容器将流量引导到 CPX DNS 容器，您需要创建一个终端节点作为 NetScaler CPX 容器的 Kubernetes 服务。此服务的群集 IP 用作应用程序 pod 的 DNS 端点。为了确保应用程序容器使用 NetScaler CPX 服务群集 IP 地址进行 DNS 解析，您需要使用 NetScaler CPX 服务群集 IP 更新每个节点上的 kubelet 配置文件。

引入了以下环境变量以支持部署 NetScaler CPX 作为 NodeLocal DNS 缓存：

- **KUBE_DNS_SVC_IP**: 指定 `kube-dns` 服务的群集 IP 地址，这是在 NetScaler CPX 容器上触发配置的必填参数。当 NetScaler CPX 缓存中没有 DNS 查询响应时，NetScaler CPX 容器会将 DNS 查询定向到此 IP 地址。
- **CPX_DNS_SVC_IP**: 指定 NetScaler CPX 服务的群集 IP 地址。`CPX_DNS_SVC_IP` 环境变量用于在节点上配置本地 DNS。配置此变量时，会添加一条 `iptables` 规则，将来自应用程序容器的 DNS 请求定向到节点内的本地 NetScaler CPX 容器。
- **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 接收的亦如此。
- **NS_DNS_EXT_RESLV_IP**: 指定外部名称服务器的 IP 地址，以定向特定域的 DNS 请求。
- **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。

将 **NetScaler CPX** 作为 **DNS** 缓存部署在节点上

部署 NetScaler CPX 作为 Kubernetes 群集的本地 DNS 缓存包括以下任务：

在主节点上：

- 创建以端点作为 NetScaler CPX 容器的 Kubernetes 服务
- 创建 ConfigMap 用于为 NetScaler CPX 容器定义环境变量
- 使用 Kubernetes DaemonSet 在 Kubernetes 群集中的每个节点上安排 NetScaler CPX 容器。

在工作节点上：

- 使用 NetScaler CPX 服务的群集 IP 地址修改 kubelet 配置文件，将 DNS 请求转发给 NetScaler CPX。

Kubernetes 主节点上的配置

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

1. 使用 `cpx_dns_svc.yaml` 文件创建以 NetScaler CPX 容器作为端点的服务。

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

`cpx_dns_svc.yaml` 文件按如下所示提供：

```
1     apiVersion: v1
2     kind: Service
3     metadata:
4       name: cpx-dns-svc
5       labels:
6         app: cpxd
7     spec:
8       ports:
9         - protocol: UDP
```



```

10         port: 53
11         name: dns
12         - protocol: TCP
13         port: 53
14         name: dns-tcp
15         selector:
16         app: cpx-daemon

```

2. 获取 NetScaler CPX 服务的 IP 地址。

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

3. 获取 Kube DNS 服务的 IP 地址。

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

4. 创建 ConfigMap 用于为 NetScaler CPX 容器定义环境变量。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，使用在文件中指定为数据（键-值对）的环境变量创建了一个示例 ConfigMap `cpx-dns-cache`。

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

下面是一个包含环境变量作为键-值对的示例文件。

```

1 CPX_DNS_SVC_IP: 10.111.95.145
2 EULA: "yes"
3 KUBE_DNS_SVC_IP: 10.96.0.10
4 NS_CPX_LITE: "1"
5 NS_DNS_EXT_RESLV_IP: 10.102.217.142
6 NS_DNS_MATCH_DOMAIN: citrix.com
7 PLATFORM: CP1000

```

下面是一个示例 ConfigMap:

```

1 apiVersion: v1
2 data:
3   CPX_DNS_SVC_IP: 10.111.95.145
4   EULA: "yes"
5   KUBE_DNS_SVC_IP: 10.96.0.10
6   NS_CPX_LITE: "1"
7   NS_DNS_EXT_RESLV_IP: 10.102.217.142
8   NS_DNS_MATCH_DOMAIN: citrix.com
9   PLATFORM: CP1000
10 kind: ConfigMap
11 metadata:
12   creationTimestamp: "2019-10-15T07:45:54Z"
13   name: cpx-dns-cache
14   namespace: default
15   resourceVersion: "8026537"
16   selfLink: /api/v1/namespaces/default/configmaps/cpx-dns-cache
17   uid: 8d06f6ee-133b-4e1a-913c-9963cbf4f48

```

5. 在主节点上为 NetScaler CPX 创建 Kubernetes 守护程序集。

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

cpx_daemonset.yaml 文件按如下所示提供：

```
1  apiVersion: apps/v1
2  kind: DaemonSet
3  metadata:
4    name: cpx-daemon
5    labels:
6      app: cpxd
7  spec:
8    selector:
9      matchLabels:
10     app: cpx-daemon
11  template:
12    metadata:
13      labels:
14        app: cpx-daemon
15    spec:
16      containers:
17      - name: cpxd
18        imagePullPolicy: IfNotPresent
19        image: localhost:5000/dev/cpx
20        volumeMounts:
21        - mountPath: /netns/default/
22          name: test-vol
23        ports:
24        - containerPort: 53
25      envFrom:
26      - configMapRef:
27        name: cpx-dns-cache
28      securityContext:
29        privileged: true
30        allowPrivilegeEscalation: true
31      capabilities:
32        add: ["NET_ADMIN"]
33      volumes:
34      - name: test-vol
35        hostPath:
36          path: /proc/1/ns
37          type: Directory
```

Kubernetes 群集中的工作节点上的配置

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，以便应用程序容器可以使用以下步骤之一使用 NetScaler CPX 服务群集 IP 进行 DNS 解析：

- 按照重新配置节点的 `kubelet` 中的步骤进行操作，然后按照以下格式修改 `--cluster-dns` 参数值。

```
1 --cluster-dns=<CPX_DNS_SVC_IP>,<KUBE_DNS_SVC_IP>
```

或

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

- 编辑 `kubelet` 配置，为 `--cluster-dns` 参数指定 NetScaler CPX 服务的群集 IP 地址和 `kube-dns` 服务 IP 地址。

```
1 root@node:~# cat /etc/systemd/system/kubelet.service.d/10-
  kubeadm.conf | grep KUBELET\_DNS\_ARGS
2
3 Environment="KUBELET_DNS_ARGS=--cluster-dns
  =10.111.95.145,10.96.0.10 --cluster-domain=cluster.
  local"
4 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS
  $KUBELET_CONFIG_ARGS $KUBELET_DNS_ARGS
```

- 使用以下命令重新加载节点的 `kubelet`:

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

在 Google Compute Engine 上部署 NetScaler CPX 代理

March 25, 2024

本部署指南描述了如何在谷歌云的谷歌计算引擎 (GCE) 上部署带有 Docker 的 NetScaler CPX，同时在企业网络中运行 NetScaler ADM。在此部署中，安装在 GCE 上的 NetScaler CPX 对两台后端服务器进行负载均衡，NetScaler ADM 提供许可和分析解决方案。

NetScaler CPX 是基于容器的代理，支持完整的 7 层功能、SSL 卸载、多个协议和 NITRO API。NetScaler ADM 提供管理、许可和分析解决方案。作为许可服务器，NetScaler ADM 为在本地或云端运行的 NetScaler CPX 实例提供授权。

CPX 和 CPX Express 是相同的映像。使用 Citrix ADM 许可并安装 CPX 映像时，Docker App Store (版本 11 或 12) 中的 CPX 映像将成为完整的 CPX 实例。如果没有许可证，CPX 映像将成为支持 20 Mbps 和 250 个 SSL 连接的 CPX Express 实例。

必备条件

- 专用于 NetScaler CPX 的 2 GB 内存和 1 个 vCPU

- 可从 GCE 获得的 Docker 开放源
- 本地运行的 Citrix ADM，具有与 GCE 的 Internet 或 VPN 连接

注意

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

配置步骤

要配置此部署，必须执行以下步骤。

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 CPX 实例。
5. 通过 NetScaler ADM 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 以实现负载均衡，并验证两个 Web 服务的负载分配。

步骤 1: 在 GCE VM 上安装 Docker

从 GCE 创建 Linux Ubuntu VM。然后，使用以下示例中所示的命令在 VM 上安装 Docker:

```

1 $ sudo curl -ssl https://get.docker.com/ | sh
2 % Total % Received % Xferd Average Speed Time Time Time Current
3 Dload Upload Total Spent Left Speed
4 0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve
   host: xn--ssl-1n0a
5 100 17409 100 17409 0 0 21510 0 --:--:-- --:--:-- --:--:-- 21492
6 apparmor is enabled in the kernel and apparmor utils were already
   installed
7 + sudo -E sh -c apt-key add -
8 + echo -----BEGIN PGP PUBLIC KEY BLOCK-----
9 Version: GnuPG v1
10
11 mQINBFWln24BEADrBl5p99uKh8+rpvqJ48u4eTtjeXAWbslJotmC/CakbNSq0b9o
12 ddfzRvGveJVERT/Q/mlvEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
13 mYdoe/BJ/J/6rH0jq70mk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
14 TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyU/jPpWFK28TR8yfVlzYFwibj5WK
15 dHM7ZTqlA1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccyOi+ULU0z8mHUJ3iEMIrPT
16 X+80KaN/ZjibfsB0CjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
17 HOoyC6qVe0gKrRiKxzymj0FIMLru/iFF5pSWcQB7PYlt8J0G80lAcPr6VCiN+4c

```

```

18 NKv03SdvA69dC0j79Pu09IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHhM1fDeCqkZ
19 hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Zaltn0svoGs+wX3Z/H6/8P5anog43U
20 65c0A+64Jj00rNDR8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+c0MHUU4HA7iM
21 zHrouAdYeTZeZEQ0A7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFNcTKdv+rwARAQAB
22 tDdEb2NrZXIgLmVsZWZzZSBub29sICHyZWxlyXNlZG9ja2VyKSA8ZG9ja2VyQGRv
23 Y2tldi5jb20+iQIcBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4Syo0CV
24 cnybl18vzwBEcp4VCRbXvHvOXty1gccVIV8/aJqNKgBV97LY3vrp0yiIeB8ETQeg
25 srxFE7t/Gz0rsL0bqfLEHdmn5iBJRkhLFcpzje0nyB3Z0IJB6Uog0/msQVYe5CXJ
26 l6uwr0AmoicBLrVLDaktxVh9RWch0l0KZR2FpHu8h+uM0/zySqIdlyfLa3y5oH
27 scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmCESyAb4vvdcaHR+bXhA/RW8QHeeMfliWw
28 7Z2jYHyuHmDnWG2yUrnCqAJTrWV+OfKRIZzJFBs4e88ru5h2ZIXdRepw/+COYj34
29 LyzXR2cxr2u/xvxwXcKsMe7F4KZaphD+1ws61FhnUMi/PERMYFTFuvPrCkq4gyBj
30 t3fFpZ2NR/fKW87Q0eVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsrxkFQ2jNp
31 D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaH0
32 W1zblWy/CFr6VfL963jrjJgag0G6tNtBZLrclZgWh0QpeZZ5Lbvz2ZA5CqRrFAVc
33 wPNW1f0bFIRtqV6vuVluFOPCMAAnOnqR02w9t17iVQj03oVN0mbQi9vjuExXh1Yo
34 ScVeti06LSmlQfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBLAAoJEFyzYeVS
35 +w0QeH0QAI6btAfYwYPuAjFRUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+l/R08nu0d
36 sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
37 NydGFzmeOKbAlrxIMECnSTG/TkFVY09Ntlv9vSN2BupmTagTRErxLZKnVsWRzp+X
38
39 -----END PGP PUBLIC KEY BLOCK-----
40
41 OK
42 + sudo -E sh -c mkdir -p /etc/apt/sources.list.d
43 + dpkg --print-architecture
44 + sudo -E sh -c echo deb \[\[arch=amd64\]\] https://apt.dockerproject.org
45   /repo ubuntu-yakkety main > /etc/apt/sources.list.d/docker.list
46 + sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-
47   engine
48 Hit:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety InRelease
49 Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates
50   InRelease [102 kB]
51 Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports
52   InRelease [102 kB]
53 Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted
54   Sources [5,376 B]
55 Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse
56   Sources [181 kB]
57 Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
58   Sources [8,044 kB]
59 Get:7 http://archive.canonical.com/ubuntu yakkety InRelease [11.5 kB]
60 Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease [102
61   kB]
62 Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease [47.3
63   kB]
64 Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main
65   Sources [903 kB]
66 Get:11 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
67   restricted Sources [2,688 B]
68 Get:12 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
69   universe Sources [57.9 kB]
70 Get:13 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/

```

```
    multiverse Sources [3,172 B]
59 Get:14 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main Sources [107 kB]
60 Get:15 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main amd64 Packages [268 kB]
61 Get:16 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main Translation-en [122 kB]
62 Get:17 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    universe amd64 Packages [164 kB]
63 Get:18 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    universe Translation-en [92.4 kB]
64 Get:19 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    multiverse amd64 Packages [4,840 B]
65 Get:20 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    multiverse Translation-en [2,708 B]
66 Get:21 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe Sources [2,468 B]
67 Get:22 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    main Sources [2,480 B]
68 Get:23 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    main amd64 Packages [3,500 B]
69 Get:24 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe amd64 Packages [3,820 B]
70 Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe Translation-en [1,592 B]
71 Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64
    Packages [2,480 B]
72 Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources
    [47.7 kB]
73 Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
    Packages [2,453 B]
74 Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe
    Sources [20.7 kB]
75 Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
    Sources [1,140 B]
76 Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted
    Sources [2,292 B]
77 Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64
    Packages [150 kB]
78 Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main
    Translation-en [68.0 kB]
79 Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe
    amd64 Packages [77.2 kB]
80 Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe
    Translation-en [47.3 kB]
81 Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
    amd64 Packages [2,832 B]
82 Fetched 10.8 MB in 2s (4,206 kB/s)
83 Reading package lists... Done
84 Reading package lists...
85 Building dependency tree...
86 Reading state information...
87 The following additional packages will be installed:
```

```
88 aufs-tools cgroupfs-mount libltdl7
89 The following NEW packages will be installed:
90 aufs-tools cgroupfs-mount docker-engine libltdl7
91 0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.
92 Need to get 21.2 MB of archives.
93 After this operation, 111 MB of additional disk space will be used.
94 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]
95 Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
    amd64 cgroupfs-mount all 1.3 [5,778 B]
96 Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main amd64
    libltdl7 amd64 2.4.6-1 [38.6 kB]
97 Get:4 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
    docker-engine amd64 17.05.0~ce-0~ubuntu-yakkety [21.1 MB]
98 Fetched 21.2 MB in 1s (19.8 MB/s)
99 Selecting previously unselected package aufs-tools.
100 (Reading database ... 63593 files and directories currently installed.)
101 Preparing to unpack .../aufs-tools_1%3a3.2+20130722-1.1ubuntu1_amd64.
    deb ...
102 Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
103 Selecting previously unselected package cgroupfs-mount.
104 Preparing to unpack .../cgroupfs-mount_1.3_all.deb ...
105 Unpacking cgroupfs-mount (1.3) ...
106 Selecting previously unselected package libltdl7:amd64.
107 Preparing to unpack .../libltdl7_2.4.6-1_amd64.deb ...
108 Unpacking libltdl7:amd64 (2.4.6-1) ...
109 Selecting previously unselected package docker-engine.
110 Preparing to unpack .../docker-engine_17.05.0~ce-0~ubuntu-yakkety_amd64
    .deb ...
111 Unpacking docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
112 Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
113 Processing triggers for ureadahead (0.100.0-19) ...
114 Setting up cgroupfs-mount (1.3) ...
115 Processing triggers for libc-bin (2.24-3ubuntu2) ...
116 Processing triggers for systemd (231-9ubuntu4) ...
117 Setting up libltdl7:amd64 (2.4.6-1) ...
118 Processing triggers for man-db (2.7.5-1) ...
119 Setting up docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
120 Created symlink /etc/systemd/system/multi-user.target.wants/docker.
    service → /lib/systemd/system/docker.service.
121 Created symlink /etc/systemd/system/sockets.target.wants/docker.socket
    → /lib/systemd/system/docker.socket.
122 Processing triggers for ureadahead (0.100.0-19) ...
123 Processing triggers for libc-bin (2.24-3ubuntu2) ...
124 Processing triggers for systemd (231-9ubuntu4) ...
125 + sudo -E sh -c docker version
126 Client:
127 Version: 17.05.0-ce
128 API version: 1.29
129 Go version: go1.7.5
130 Git commit: 89658be
131 Built: Thu May 4 22:15:36 2017
132 OS/Arch: linux/amd64
```

```
133
134 Server:
135 Version: 17.05.0-ce
136 API version: 1.29 (minimum version 1.12)
137 Go version: go1.7.5
138 Git commit: 89658be
139 Built: Thu May 4 22:15:36 2017
140 OS/Arch: linux/amd64
141 Experimental: false
142
143 If you would like to use Docker as a non-root user, you should now
144 consider
145 adding your user to the "docker" group with something like:
146 sudo usermod -aG docker albert_lee
147
148 Remember that you will have to log out and back in for this to take
149 effect.
150
151 WARNING: Adding a user to the "docker" group will grant the ability to
152 run
153 containers which can be used to obtain root privileges on the
154 docker host.
155 Refer to https://docs.docker.com/engine/security/security/#docker-
156 daemon-attack-surface
157 for more information.
158 $
159 \*\*$ sudo docker info\*\*
160 Containers: 0
161 Running: 0
162 Paused: 0
163 Stopped: 0
164 Images: 0
165 Server Version: 17.05.0-ce
166 Storage Driver: aufs
167 Root Dir: /var/lib/docker/aufs
168 Backing Filesystem: extfs
169 Dirs: 0
170 Dirperm1 Supported: true
171 Logging Driver: json-file
172 Cgroup Driver: cgroupfs
173 Plugins:
174 Volume: local
175 Network: bridge host macvlan null overlay
176 Swarm: inactive
177 Runtimes: runc
178 Default Runtime: runc
179 Init Binary: docker-init
180 containerd version: 9048e5e50717ea4497b757314bad98ea3763c145
181 runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228
182 init version: 949e6fa
```



```

182 Security Options:
183 apparmor
184 seccomp
185 Profile: default
186 Kernel Version: 4.8.0-51-generic
187 Operating System: Ubuntu 16.10
188 OSType: linux
189 Architecture: x86_64
190 CPUs: 1
191 Total Memory: 3.613GiB
192 Name: docker-7
193 ID: R5TW:VKXK:EKGR:GHWM:UNU4:LPJH:IQY5:X77G:NNRQ:HWBY:LIUD:4ELQ
194 Docker Root Dir: /var/lib/docker
195 Debug Mode (client): false
196 Debug Mode (server): false
197 Registry: https://index.docker.io/v1/
198 Experimental: false
199 Insecure Registries:
200 127.0.0.0/8
201 Live Restore Enabled: false
202
203 WARNING: No swap limit support
204 $
205
206 \*\*$ sudo docker images\*\*
207 REPOSITORY TAG IMAGE ID CREATED SIZE
208 $
209
210 \*\*$ sudo docker ps\*\*
211 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
212 $
213 <!--NeedCopy-->

```

步骤 2: 配置与 **Docker** 实例的远程 **API** 通信

打开端口 4243 用于与 Docker 实例的 API 通信。NetScaler ADM 需要此端口才能与 Docker 实例进行通信。

```

1
2 \*\*cd /etc/systemd/system\*\*
3 \*\*sudo vi docker-tcp.socket\*\*
4 \*\*cat docker-tcp.socket\*\*
5 [Unit]
6 \*\*Description=Docker Socket for the API
7 [Socket]
8 ListenStream=4243
9 BindIPv6Only=both
10 Service=docker.service
11 [Install]
12 WantedBy=sockets.target\*\*
13
14 $ \*\*sudo systemctl enable docker-tcp.socket\*\*

```

```
15 Created symlink /etc/systemd/system/sockets.target.wants/docker-tcp.
    socket → /etc/systemd/system/docker-tcp.socket.
16 \*\*sudo systemctl enable docker.socket\*\*
17 \*\*sudo systemctl stop docker\*\*
18 \*\*sudo systemctl start docker-tcp.socket\*\*
19 \*\*sudo systemctl start docker\*\*
20 $ \*\*sudo systemctl status docker\*\*
21 ● docker.service - Docker Application Container Engine
22 Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
    preset: enabled)
23 Active: \*\*active (running)\*\* since Wed 2017-05-31 12:52:17 UTC; 2s
    ago
24 Docs: https://docs.docker.com
25 Main PID: 4133 (dockerd)
26 Tasks: 16 (limit: 4915)
27 Memory: 30.1M
28 CPU: 184ms
29 CGroup: /system.slice/docker.service
30 └─4133 /usr/bin/dockerd -H fd://
31 └─4137 docker-containerd -l unix:///var/run/docker/libcontainerd/docker
    -containerd.sock --metrics-interval=0 --start-timeout 2m -
32
33 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
34 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
35 May 31 12:52:17 docker-7 dockerd[4133]: time="2017-05-31T12
    :52:17.301681794Z" level=info msg="Loading containers: start."
36 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
37 May 31 12:52:17 docker-7 dockerd[4133]: time="2017-05-31T12
    :52:17.465011600Z" level=info msg="Loading containers: done."
38 May 31 12:52:17 docker-7 dockerd[4133]: time="2017-05-31T12
    :52:17.484747909Z" level=info msg="Daemon has completed
    initialization"
39 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
40 May 31 12:52:17 docker-7 systemd[1]: Started Docker Application
    Container Engine.
41 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"
42 May 31 12:52:17 docker-7 dockerd[4133]: time="2017-05-31T12
    :52:17.504061522Z" level=info msg="API listen on [::]:4243"
43 $
44
45 (external)$ \*\*curl 104.199.209.157:4243/version\*\*
46 {
47   "Version":"17.05.0-ce","ApiVersion":"1.29","MinAPIVersion":"1.12","
```

```

    GitCommit":"89658be","GoVersion":"go1.7.5","Os":"linux","Arch":"
    amd64","KernelVersion":"4.8.0-52-generic","BuildTime":"2017-05-04
    T22:15:36.071254972+00:00" }
48
49 (external)$
50
51 <!--NeedCopy-->

```

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 Citrix ADM 许可并安装 CPX 映像时，该映像将成为具有 1 Gbps 性能的完整 CPX 实例。如果没有许可证，该映像将成为支持 20 Mbps 和 250 个 SSL 连接的 CPX Express 实例。

```

1 $ \*\sudo docker pull store/citrix/citrixadccpx:13.0-36.29\*\*
2 13.0-36.29: Pulling from store/citrix/citrixadccpx
3 4e1f679e8ab4: Pull complete
4 a3ed95caeb02: Pull complete
5 2931a926d44b: Pull complete
6 362cd40c5745: Pull complete
7 d10118725a7a: Pull complete
8 1e570419a7e5: Pull complete
9 d19e06114233: Pull complete
10 d3230f008ffd: Pull complete
11 22bdb10a70ec: Pull complete
12 1a5183d7324d: Pull complete
13 241868d4ebff: Pull complete
14 3f963e7ae2fc: Pull complete
15 fd254cf1ea7c: Pull complete
16 33689c749176: Pull complete
17 59c27bad28f5: Pull complete
18 588f5003e10f: Pull complete
19 Digest: sha256:31
    a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe
20 Status: Downloaded newer image for store/citrix/citrixadccpx:13.0-36.29
21 $
22
23 $ \*\sudo docker images\*\*
24 REPOSITORY TAG IMAGE ID CREATED SIZE
25 store/citrix/citrixadccpx:13.0-36.29 6fa57c38803f 3 weeks ago 415MB
26 $
27 <!--NeedCopy-->

```

步骤 4: 创建 NetScaler CPX 实例

在 Docker 主机上安装 NetScaler CPX 映像。打开用于特定服务的端口（如下面的示例中所示），并为 NetScaler ADM 指定 IP 地址：

```

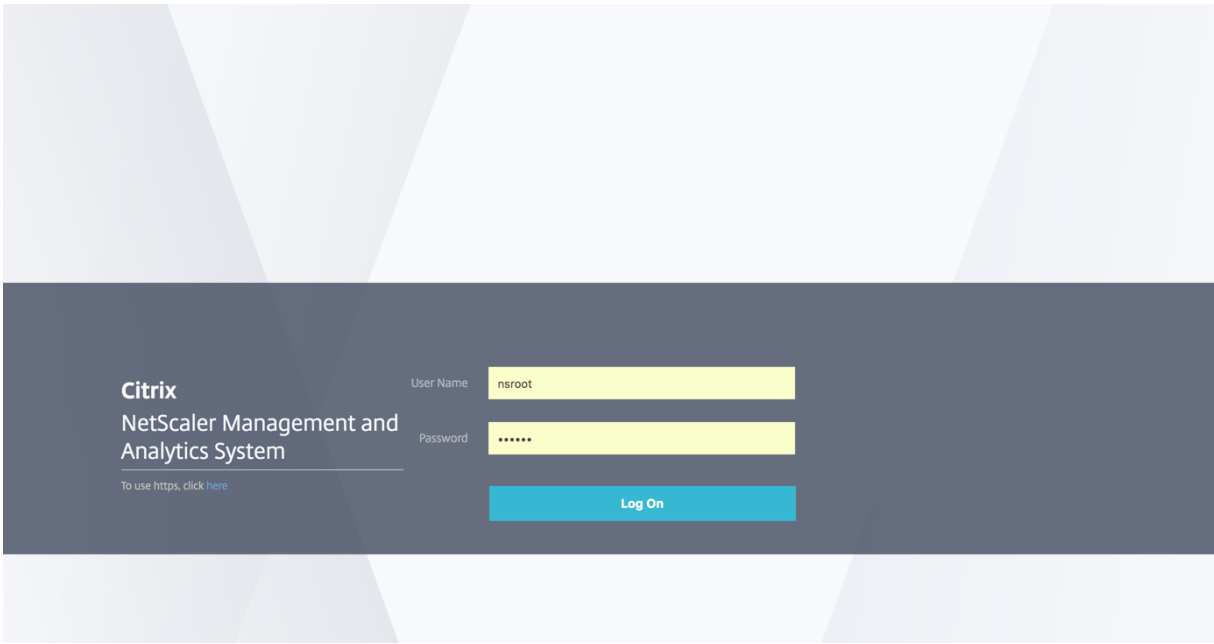
1 bash-2.05b# \*\*CHOST=${
2 1:-localhost }
3 \*\*
4 bash-2.05b# \*\*echo | openssl s_client -connect $CHOST:443 | openssl
   x509 -fingerprint -noout | cut -d'=' -f2\*\*
5 depth=0 C = US, ST = California, L = San Jose, O = Citrix NetScaler, OU
   = Internal, CN = Test Only Cert
6 verify error:num=18:self signed certificate
7 verify return:1
8 depth=0 C = US, ST = California, L = San Jose, O = Citrix NetScaler, OU
   = Internal, CN = Test Only Cert
9 verify return:1
10 DONE
11 24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4
12 bash-2.05b#
13
14 $ \*\*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\*\*
15 44ca1c6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
16 $
17
18 $ \*\*sudo docker ps\*\*
19 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
20 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_perلمان
21 $
22
23 $ \*\*ssh -p 5022 root@localhost\*\*
24 root@localhost's password:
25 Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86_64)
26
27 * Documentation: https://www.citrix.com/
28 Last login: Mon Jun 5 18:58:51 2017 from xx.xx.xx.xx
29 root@44ca1c6c0907:~#
30 root@44ca1c6c0907:~#
31 root@44ca1c6c0907:~# \*\*cli_script.sh 'show ns ip'\*\*
32 exec: show ns ip
33 Ippaddress Traffic Domain Type Mode Arp Icmp Vserver State
34 -----
35 1) 172.17.0.2 0 NetScaler IP Active Enabled Enabled NA Enabled
36 2) 192.0.0.1 0 SNIP Active Enabled Enabled NA Enabled
37 Done
38 root@44ca1c6c0907:~# \*\*cli_script.sh 'show licenseserver'\*\*
39 exec: show licenseserver
40 1) ServerName: xx.xx.xx.xxPort: 27000 Status: 1 Grace: 0 Gptimeleft: 0

```

```
41 Done
42 root@44ca1c6c0907:~# cli_script.sh 'show capacity'
43 exec: show capacity
44 Actualbandwidth: 1000 Platform: CP1000 Unit: Mbps Maxbandwidth: 3000
   Minbandwidth: 20 Instancecount: 0
45 Done
46 root@44ca1c6c0907:~#
47
48 $ \*\*sudo iptables -t nat -L -n\*\*
49 Chain PREROUTING (policy ACCEPT)
50 target prot opt source destination
51 DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
52
53 Chain INPUT (policy ACCEPT)
54 target prot opt source destination
55
56 Chain OUTPUT (policy ACCEPT)
57 target prot opt source destination
58 DOCKER all -- 0.0.0.0/0 !127.0.0.0/8 ADDRTYPE match dst-type LOCAL
59
60 Chain POSTROUTING (policy ACCEPT)
61 target prot opt source destination
62 MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
63 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
64 MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
65 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
66 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
67 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
68
69 Chain DOCKER (2 references)
70 target prot opt source destination
71 RETURN all -- 0.0.0.0/0 0.0.0.0/0
72 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
73 DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
74 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
75 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
76 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
77 $
78 <!--NeedCopy-->
```

第 5 步: 通过 NetScaler ADM 许可 NetScaler CPX

假设 NetScaler ADM 在本地运行, 您应该能够验证 NetScaler CPX 是否在与 MAS 通信并发送信息。下图显示 NetScaler CPX 正在从 NetScaler ADM 取回许可证。



Citrix NetScaler Management and Analytics System May 31 2017 13:14:20 GMT nsroot

Search here

Networks > License Settings

License Server Port Settings

Proxy Server Port	License Server Port	Vendor Daemon Port
0	27000	7279

License Files

The following license files are present on this server. Select **Add New License** to upload more licenses. To delete a license, select the license and click **Delete**.

<input type="checkbox"/>	Name	Last Modified	Size
<input type="checkbox"/>	FID_3bf0b423_15ba7640cc6_2664.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_3bf0b423_15ba7640cc6_2672.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_2a2386a8_15b93284902_487e.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_2a2386a8_15b93284902_4878.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_3bf0b423_15ba7640cc6_5281.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_2a2386a8_15b93284902_4870.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_3bf0b423_15ba7640cc6_527b.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_2a2386a8_15b93284902_486a.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_3bf0b423_15ba7640cc6_5275.lic	2017-05-23 21:30:15	1.10 KB
<input type="checkbox"/>	FID_2a2386a8_15b93284902_4864.lic	2017-05-23 21:30:15	1.10 KB

License Expiry Information

Feature	Count	Days To Expiry
No Items		

Citrix NetScaler Management and Analytics System Jun 05 2017 15:09:41 GMT nsroot

Search here

Networks > Instances > NetScaler CPX

NetScaler CPX

Instances **3** Docker Host **0**

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

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 上配置负载均衡以对两台 Web 服务器进行负载均衡，然后测试配置。

安装 NGINX Web 服务器 使用以下示例中所示的命令安装 NGINX Web 服务器。

```

1 $ sudo docker pull nginx
2 Using default tag: latest
3 latest: Pulling from library/nginx
4 Digest: sha256:41
   ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
5 Status: Image is up to date for nginx:latest
6
7
8 \*\*$ sudo docker run -d -p 81:80 nginx\*\*
9 098a77974818f451c052ecd172080a7d45e446239479d9213cd4ea6a3678616f
10
11
12 \*\*$ sudo docker run -d -p 82:80 nginx\*\*
13 bbdac2920bb4085f70b588292697813e5975389dd546c0512daf45079798db65
14
15
16 \*\*$ sudo iptables -t nat -L -n\*\*
17 Chain PREROUTING (policy ACCEPT)
18 target prot opt source destination
19 DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
20
21 Chain INPUT (policy ACCEPT)
22 target prot opt source destination
23
24 Chain OUTPUT (policy ACCEPT)
25 target prot opt source destination
26 DOCKER all -- 0.0.0.0/0 !127.0.0.0/8 ADDRTYPE match dst-type LOCAL
27

```

```

28 Chain POSTROUTING (policy ACCEPT)
29 target prot opt source destination
30 MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
31 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
32 MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
33 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
34 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
35 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
36 MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80
37 MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80
38
39 Chain DOCKER (2 references)
40 target prot opt source destination
41 RETURN all -- 0.0.0.0/0 0.0.0.0/0
42 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
43 DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
44 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
45 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
46 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
47 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80
48 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80
49 $
50 <!--NeedCopy-->

```

配置 **NetScaler CPX** 以实现负载均衡并验证两个 **Web** 服务的负载均衡

```

1 $ \*\ssh -p 5022 root@localhost\*\*
2 root@localhost's password:
3 Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86_64)
4
5 * Documentation: https://www.citrix.com/
6 Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1
7 root@44ca1c6c0907:~#
8 root@44ca1c6c0907:~#
9 root@44ca1c6c0907:~#
10 root@44ca1c6c0907:~#
11 root@44ca1c6c0907:~# \*\cli_script.sh "add service web1 172.17.0.3
    HTTP 80"\*\*
12 exec: add service web1 172.17.0.3 HTTP 80
13 Done
14 root@44ca1c6c0907:~# \*\cli_script.sh "add service web2 172.17.0.4
    HTTP 80"\*\*
15 exec: add service web2 172.17.0.4 HTTP 80
16 Done
17 root@44ca1c6c0907:~# \*\cli_script.sh "add lb vserver cpx-vip HTTP
    172.17.0.2 88"\*\*
18 exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
19 Done
20 root@44ca1c6c0907:~# \*\cli_script.sh "bind lb vserver cpx-vip web1
    "\*\*
21 exec: bind lb vserver cpx-vip web1
22 Done
23 root@44ca1c6c0907:~# \*\cli_script.sh "bind lb vserver cpx-vip web2
    "\*\*

```



```
24 exec: bind lb vserver cpx-vip web2
25 Done
26 root@44ca1c6c0907:~#
27
28 root@44ca1c6c0907:~# \*\*cli_script.sh 'show lb vserver cpx-vip'\*\*
29 exec: show lb vserver cpx-vip
30
31 cpx-vip (172.17.0.2:88) - HTTP Type: ADDRESS
32 State: UP
33 Last state change was at Mon Jun 5 19:01:49 2017
34 Time since last state change: 0 days, 00:00:42.620
35 Effective State: UP
36 Client Idle Timeout: 180 sec
37 Down state flush: ENABLED
38 Disable Primary Vserver On Down : DISABLED
39 Appflow logging: ENABLED
40 Port Rewrite : DISABLED
41 No. of Bound Services : 2 (Total) 2 (Active)
42 Configured Method: LEASTCONNECTION
43 Current Method: Round Robin, Reason: A new service is bound
   BackupMethod: ROUNDROBIN
44 Mode: IP
45 Persistence: NONE
46 Vserver IP and Port insertion: OFF
47 Push: DISABLED Push VServer:
48 Push Multi Clients: NO
49 Push Label Rule: none
50 L2Conn: OFF
51 Skip Persistency: None
52 Listen Policy: NONE
53 IcmpResponse: PASSIVE
54 RHISate: PASSIVE
55 New Service Startup Request Rate: 0 PER_SECOND, Increment Interval: 0
56 Mac mode Retain Vlan: DISABLED
57 DBS_LB: DISABLED
58 Process Local: DISABLED
59 Traffic Domain: 0
60 TROFS Persistence honored: ENABLED
61 Retain Connections on Cluster: NO
62
63 2) web1 (172.17.0.3: 80) - HTTP State: UP Weight: 1
64 3) web2 (172.17.0.4: 80) - HTTP State: UP Weight: 1
65 Done
66 root@44ca1c6c0907:~#
67
68 (external)$ \*\*curl 104.199.209.157:50000\*\*
69 \\




```

```

76 width: 35em;
77 margin: 0 auto;
78 font-family: Tahoma, Verdana, Arial, sans-serif;
79 }
80
81 \</style\>
82 \</head\>
83 \<body\>
84 \<h1\>Welcome to nginx\!\</h1\>
85 \<p\>If you see this page, the nginx web server is successfully
    installed and
86 working. Further configuration is required.\</p\>
87
88 \<p\>For online documentation and support please refer to
89 \<a href="http://nginx.org/"\>nginx.org\</a\>.\<br/\>
90 Commercial support is available at
91 \<a href="http://nginx.com/"\>nginx.com\</a\>.\</p\>
92
93 \<p\>\<em\>Thank you for using nginx.\</em\>\</p\>
94 \</body\>
95 \</html\>
96 (external)$
97
98
99 (external)$ for i in {
100 1..100 }
101 ; \*\*do curl http://104.199.209.157:50000 -o /dev/null ; done\*\*
102
103 % Total      % Received % Xferd  Average Speed   Time    Time     Time
    Current
104
105           Speed
106
107 100   612   100   612     0     0   1767     0  --:--:--  --:--:--
    --:--:--  1768
108
109 % Total      % Received % Xferd  Average Speed   Time    Time     Time
    Current
110
111           Speed
112
113 100   612   100   612     0     0   1893     0  --:--:--  --:--:--
    --:--:--  1894
114
115 % Total      % Received % Xferd  Average Speed   Time    Time     Time
    Current
116
117           Speed
118
119 100   612   100   612     0     0   1884     0  --:--:--  --:--:--

```

120	--:--:-- 1883											
121	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
122	Current											
123				Dload	Upload	Total	Spent	Left				
124	Speed											
125	100	612	100	612	0	0	1917	0	--:--:--	--:--:--		
126	--:--:-- 1924											
127	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
128	Current											
129				Dload	Upload	Total	Spent	Left				
130	Speed											
131	100	612	100	612	0	0	1877	0	--:--:--	--:--:--		
132	--:--:-- 1883											
133	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
134	Current											
135				Dload	Upload	Total	Spent	Left				
136	Speed											
137	100	612	100	612	0	0	1852	0	--:--:--	--:--:--		
138	--:--:-- 1848											
139	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
140	Current											
141				Dload	Upload	Total	Spent	Left				
142	Speed											
143	100	612	100	612	0	0	1860	0	--:--:--	--:--:--		
144	--:~:~:~ 1865											
145	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
146	Current											
147				Dload	Upload	Total	Spent	Left				
148	Speed											
149	100	612	100	612	0	0	1887	0	--:~:~:~ 1888	--:~:~:~		
150	--:~:~:~ 1888											
151	% Total	% Received	% Xferd	Average	Speed	Time	Time	Time				
152	Current											
153				Dload	Upload	Total	Spent	Left				
154	Speed											

155	100	612	100	612	0	0	1802	0	--:--:--	--:--:--
				1800						
156										
157	% Total		% Received	% Xferd			Average Speed		Time	Time
	Current									Time
158										
159							Dload	Upload	Total	Spent
	Speed									Left
160										
161	100	612	100	612	0	0	1902	0	--:--:--	--:--:--
				1906						
162										
163	% Total		% Received	% Xferd			Average Speed		Time	Time
	Current									Time
164										
165							Dload	Upload	Total	Spent
	Speed									Left
166										
167	100	612	100	612	0	0	1843	0	--:--:--	--:--:--
				1848						
168										
169										
170										
171	% Total		% Received	% Xferd			Average Speed		Time	Time
	Current									Time
172										
173							Dload	Upload	Total	Spent
	Speed									Left
174										
175	100	612	100	612	0	0	1862	0	--:--:--	--:--:--
				1860						
176										
177	% Total		% Received	% Xferd			Average Speed		Time	Time
	Current									Time
178										
179							Dload	Upload	Total	Spent
	Speed									Left
180										
181	100	612	100	612	0	0	1806	0	--:--:--	--:--:--
				1810						
182										
183	% Total		% Received	% Xferd			Average Speed		Time	Time
	Current									Time
184										
185							Dload	Upload	Total	Spent
	Speed									Left
186										
187	100	612	100	612	0	0	1702	0	--:--:--	--:--:--
				1704						
188										
189	(external)\$									
190										
191										

```
192
193
194
195 root@44ca1c6c0907:~# \*\*cli_script.sh 'stat lb vserver cpx-vip'\*\*
196
197 exec: stat lb vserver cpx-vip
198
199
200
201 Virtual Server Summary
202
203           vsvrIP  port  Protocol  State  Health
204   actSvcs
205 cpx-vip      172.17.0.2  88      HTTP      UP      100
206           2
207
208   inactSvcs
209 cpx-vip      0
210
211
212
213
214
215 Virtual Server Statistics
216
217           Rate (/s)
218   Total
219 Vserver hits           0
220   101
221 Requests           0
222   101
223 Responses           0
224   101
225 Request bytes           0
226   8585
227 Response bytes           0
228   85850
229 Total Packets rcvd           0
230   708
231 Total Packets sent           0
232   408
233 Current client connections  --
234   0
```

234		
235	Current Client Est connections	--
	0	
236		
237	Current server connections	--
	0	
238		
239	Current Persistence Sessions	--
	0	
240		
241	Requests in surge queue	--
	0	
242		
243	Requests in vserver's surgeQ	--
	0	
244		
245	Requests in service's surgeQs	--
	0	
246		
247	Spill Over Threshold	--
	0	
248		
249	Spill Over Hits	--
	0	
250		
251	Labeled Connection	--
	0	
252		
253	Push Labeled Connection	--
	0	
254		
255	Deferred Request	0
	0	
256		
257	Invalid Request/Response	--
	0	
258		
259	Invalid Request/Response Dropped	--
	0	
260		
261	Vserver Down Backup Hits	--
	0	
262		
263	Current Multipath TCP sessions	--
	0	
264		
265	Current Multipath TCP subflows	--
	0	
266		
267	Apdex for client response times.	--
	1.00	
268		
269	Average client TTLB	--

```

270      0
271 web1      172.17.0.3      80      HTTP      UP      51
           0/s
272
273 web2      172.17.0.4      80      HTTP      UP      50
           0/s
274
275 Done
276
277 root@44ca1c6c0907:~#
278 <!--NeedCopy-->

```

NetScaler CPX 故障排除

November 23, 2023

本文档介绍如何解决在使用 NetScaler CPX 时可能遇到的问题。使用本文档，您可以收集日志以确定原因并应用变通方法来解决与 NetScaler CPX 的安装和配置有关的一些常见问题。

- 我怎样才能查看 NetScaler CPX 日志？

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

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

示例：

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

以下是带有 `tty: true` 选项的 NetScaler CPX 容器部署示例：

```

1 containers:
2   - name: cpx-ingress
3     image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-58.30"
4     tty: true
5     securityContext:
6       privileged: true
7     env:
8
9 <!--NeedCopy-->

```

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：要获取提供点名称，请运行 `kubectl get pods -o wide` 命令。

- 我怎样才能从 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来收集 NetScaler CPX 技术支持包：

```
1 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 复制到所需的目的地。

示例：

```
1 root@localhost# kubectl exec cpx-ingress1-55b9b6fc75-t5kc6 -c cpx
  -n citrix-adc /var/netscaler/bins/cli_script.sh "show
  techsupport"
2 exec: show techsupport
3 Scope: NODE
4 Done
5 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
6 root@localhost# ll /tmp/collector_P_192.168.29.232
  _31Aug2020_07_30.tar.gz
7 -rw-r--r-- 1 root root 1648109 Aug 31 13:23 /tmp/collector_P_192
  .168.29.232_31Aug2020_07_30.tar.gz
```

- 为什么 NetScaler CPX 容器在启动时卡住了？

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

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

示例：

```
1 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 容器和 Kubernetes 主节点之间复制文件？

建议使用 docker 的卷装载功能将 `/cpx` 目录装载到主机的文件系统中。如果 NetScaler CPX 容器退出核心转储，则装载点上有日志和其他重要数据可用。

您可以使用以下任何一个命令在 NetScaler CPX 容器和 Kubernetes 主节点之间复制文件：

kubectl cp：可以运行以下命令将文件从 pod 复制到节点：

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

示例：


```

1 root@localhost:~# kubectl cp cpx-ingress-596d56bb6-zbx6h:cpx/log/
  boot.log /tmp/cpx-boot.log -c cpx-ingress
2 root@localhost:~# ll /tmp/cpx-boot.log
3 -rw-r--r-- 1 root root 7880 Sep 11 00:07 /tmp/cpx-boot.log

```

scp: 您可以使用该命令在 NetScaler CPX 容器和 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。系统提示输入密码时，请为 SSH 用户提供密码：

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

示例：

```

1 root@localhost:~# scp nsroot@192.168.29.198:/cpx/log/boot.log /
  tmp/cpx-boot.log
2 nsroot@192.168.29.198's password:
3 boot.log
4 100% 7880      5.1MB/s   00:00
5 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 接口。运行以下命令启动 NetScaler CPX 容器的 shell 界面：

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

示例：

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

同时，运行以下命令开始执行数据包捕获：

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

如果要停止正在进行的数据包捕获，请运行以下命令：

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

您可以在 NetScaler CPX 文件系统的 `/cpx/nstrace/time-stamp` 目录中查看 `.cap` 文件中捕获的数据包。

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

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>` 命令来验证从 NetScaler CPX 节点到 NetScaler ADM 的连接。

如果可以从该节点访问 NetScaler ADM，则必须验证 `/cpx/log/boot.log` 文件中的许可证服务器配置日志。您还可以在 NetScaler CPX 容器的 shell 界面上使用以下命令检查许可证服务器配置：

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

示例：

```

1 root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
  licenseserver"
2 exec: show licenseserver
3 ServerName: 10.106.102.199Port: 27000 Status: 1 Grace: 0
  Gptimeleft: 720
4 Done

```

- 为什么即使在 NetScaler CPX 上成功配置了许可证服务器，仍未在 NetScaler CPX 上配置池化许可证？

验证 `/cpx/log/boot.log` 文件中的许可证配置日志。您也可以在 NetScaler CPX 容器的 shell 界面上使用以下命令在 NetScaler CPX 上验证配置的池化许可证：

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

示例：

```

1 root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
  capacity"
2 exec: show capacity
3 Actualbandwidth: 1000 MaxVcpuCount: 2 Edition: Platinum
  Unit: Mbps Bandwidth: 0 ` `Maxbandwidth: 40000
  Minbandwidth: 20 Instancecount: 1
4 Done

```

此外，请务必在许可证服务器中上载所需的许可证文件。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的可用许可证。在 NetScaler CPX pod 的 shell 界面上运行以下命令：

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

示例：

```

1 root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
  licenseserverpool"
2 exec: show licenseserverpool
3 Instance Total : 5
4 Instance Available : 4
5 Standard Bandwidth Total : 0 Mbps
6 Standard Bandwidth Availabe : 0 Mbps
7 Enterprise Bandwidth Total : 0 Mbps
8 Enterprise Bandwidth Available : 0 Mbps
9 Platinum Bandwidth Total : 10.00 Gbps
10 Platinum Bandwidth Available : 9.99 Gbps
11 CP1000 Instance Total : 100
12 CP1000 Instance Available : 100
13 Done
14 <!--NeedCopy-->

```

- 为什么 NITRO API 调用会收到 NetScaler CPX 的连接被拒绝响应？

从 NetScaler CPX 版本 12.1 起，NITRO API 的默认端口为 9080（不安全）和 9443（安全）。确保您尝试访问

的 NetScaler CPX 的 NITRO 端口已在容器上公开。您可以运行 `kubectl describe` 命令在 NetScaler CPX 容器部分中查看 NetScaler CPX 容器的公开和映射端口：

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

示例：

```
1      ng472 | grep -i port
2      Ports:          80/TCP, 443/TCP, 9080/TCP, 9443/TCP
3      Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP
4      NS_HTTP_PORT:   9080
5      NS_HTTPS_PORT:  9443
6      Port:           <none>
7      Host Port:      <none>
8      NS_PORT:        80
9      <!--NeedCopy-->
```

- 为什么即使没有流量或流量很少，NetScaler CPX 中的 NSPPE 进程也会消耗大部分 CPU 使用率？

如果使用 `CPX_CONFIG=' { "YIELD" : " NO" } '` 环境变量部署 NetScaler CPX，则即使没有流量或流量很少，NSPPE 进程也会消耗 100% 的 CPU 使用率。如果您希望 NSPPE 进程不消耗 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 CPU 使用率。

- 为什么 NetScaler CPX 没有在 NetScaler ADM 中列出，即使它是使用注册到 NetScaler ADM 所需的环境变量进行部署的？

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

您可以使用 `ping` 命令验证来自 NetScaler CPX 容器的 NetScaler ADM IP 地址的可访问性。此外，请确保为 NetScaler CPX 容器配置 NetScaler ADM 注册所需的所有环境变量。

- `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。

- 为什么在更改 nsroot 用户的密码后，`cli_script.sh` 会显示 *Invalid user name or password*（用户名或密码无效）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（即凭据）是可选的。如果 nsroot 用户的密码已更改，则需要提供凭据作为

`cli_script.sh` 的第二个参数。可以运行以下命令以使用凭据运行 NSCLI:

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

示例:

```
1 root@087a1e34642d:/# cli_script.sh "show ns ip"
2 exec: show ns ip
3
4 ERROR: Invalid username or password
5
6 root@087a1e34642d:/# cli_script.sh "show ns ip" ":nsroot:
7 nsroot123"
8 exec: show ns ip
9
10 Ipaddress      Traffic Domain      Type      Mode
11      Arp      Icmp      Vserver  State
12 -----
13 172.17.0.3      0
14   Enabled  Enabled  NA      Enabled  NetScaler IP  Active
15 192.0.0.1      0
16   Enabled  Enabled  NA      Enabled  SNIP          Active
17 Done
```

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

从 13.0-64.35 版本起, NetScaler CPX 会生成默认密码并针对 SSH 用户 (`root` 和 `nsroot`) 进行更新。如果您没有手动更改密码, 则可以在 NetScaler CPX 的文件系统的 `/var/deviceinfo/random_id` 中找到 SSH 用户的密码。



© 2024 Cloud Software Group, Inc. All rights reserved. Cloud Software Group, the Cloud Software Group logo, and other marks appearing herein are property of Cloud Software Group, Inc. and/or one or more of its subsidiaries, and may be registered with the U.S. Patent and Trademark Office and in other countries. All other marks are the property of their respective owner(s).
