Citrix ADC CPX 13.1

Contents

关于 Citrix ADC CPX

February 21, 2022

Citrix ADC CPX 是基于容器的应用程序交付控制器，可以在 Docker 主机上置备。通过 Citrix ADC CPX，客户可以利用 Docker 引擎功能，并将 Citrix 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 Citrix ADC CPX 实例作为独立的实例在 Docker 主机上进行部署。

一个 Citrix ADC CPX 实例提供高达 1 Gbps 的吞吐量。

作为 Citrix ADC 的容器化外形规格，Citrix ADC CPX 可以很好地集成到 Kubernetes 环境中，并构成 Citrix 云原生解决方案的组成部分。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

Citrix ADC 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
  - 浪涌保护和优先级队列
  - 动态路由协议
- 应用程序加速
  - 客户端和服务器 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)。Citrix ADC CPX 支持其他 Citrix 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
  - Citrix 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：
>
> 为分配给 Citrix ADC CPX 设备的接口（Linux 主机）禁用了 Rx、Tx、GRO、GSO 和 LRO 等接口功能。即使在 Citrix ADC 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 Citrix ADC CPX：

- 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)
- Docker 版本 1.12 及更高版本

体系结构和通信流

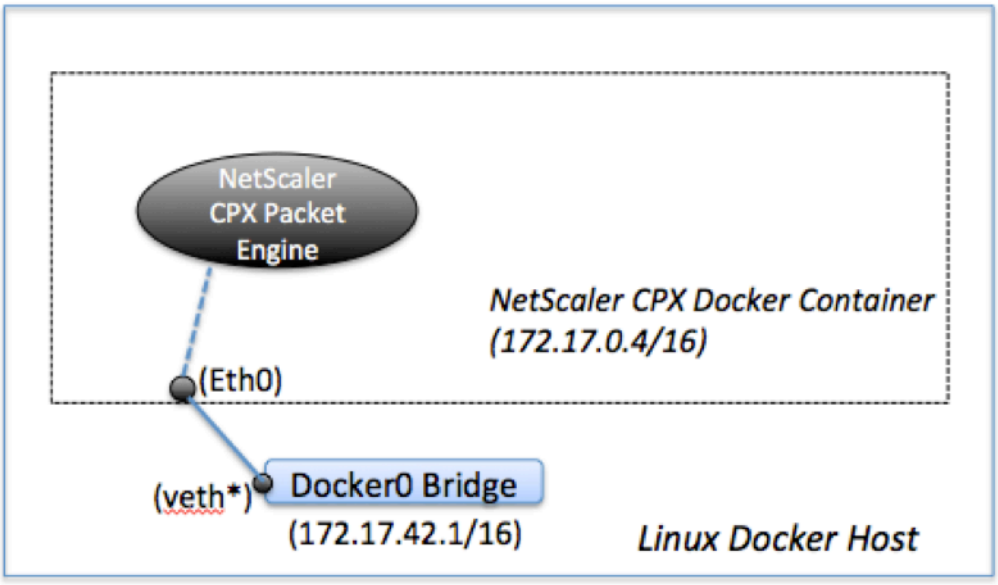
September 7, 2021

本部分内容介绍 Citrix ADC CPX 桥接模式体系结构和通信流。Citrix ADC CPX 也可以在主机模式下部署。

在 Docker 主机上预配 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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 Citrix ADC CPX 实例上的 eth0 接口接收，并由 Citrix ADC CPX 数据包引擎处理。

下图说明了 Docker 主机上的 Citrix ADC CPX 实例的体系结构。



单个 **IP** 地址在 **Citrix ADC CPX** 上的工作原理

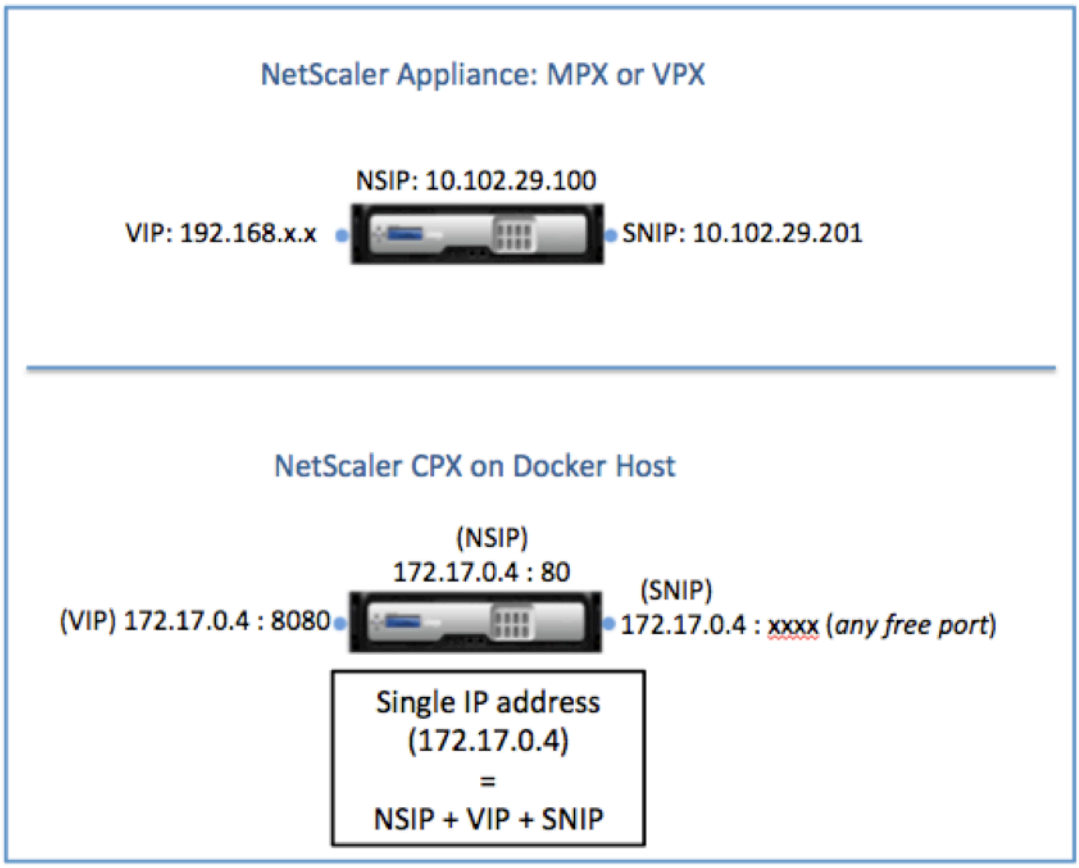
常规 Citrix ADC MPX 或 VPX 设备至少需要三个 IP 地址才能正常工作：

- 名为 Citrix 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

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

预配过程中，Docker 引擎只为一个 Citrix ADC CPX 实例分配一个专用 IP 地址（单一 IP 地址）。Citrix 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。

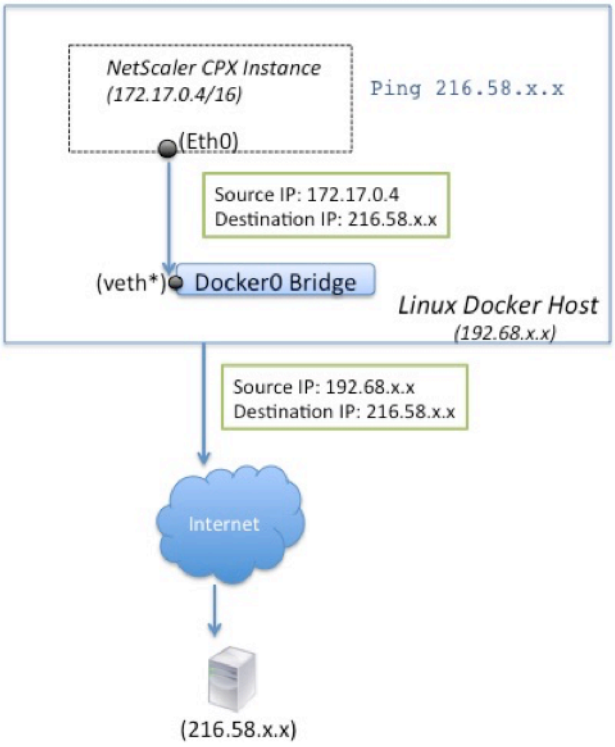
下图说明如何使用单一 IP 地址来执行 NSIP、SNIP 和 VIP 的功能。



源自 **Citrix ADC 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 Citrix ADC CPX 实例的流量导向至 docker0 IP 地址。

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



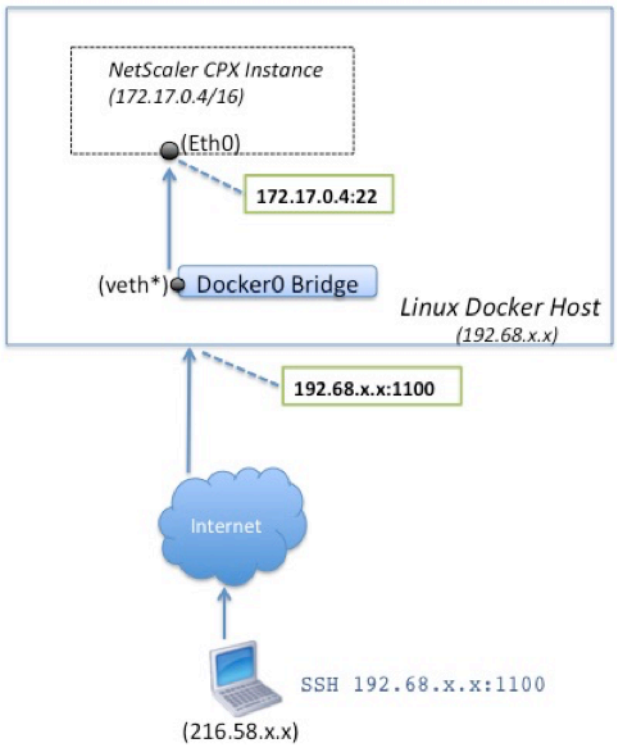
在此示例中，由 eth0 接口上的数据包引擎发送 ping 请求，来源 IP 地址为 Citrix ADC CPX IP 地址 (172.17.0.4)。然后 Docker 主机执行网络地址转换 (NAT) 以将主机 IP 地址 (192.68.x.x) 添加为来源 IP 地址，并将请求发送至目标 (216.58.x.x)。反之，来自目标 IP 地址的响应使用相同的路径。Docker 主机对响应执行 NAT 并将响应转发至 Citrix ADC CPX 实例的 eth0 接口上。

源自外部网络的请求的通信流

为了能够进行外部通信，预配 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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，Citrix ADC CPX 实例的单一 IP 地址是 172.17.0.4。Citrix ADC 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。

Citrix ADC VPX 许可

February 21, 2022

Citrix ADC CPX 是基于容器的应用程序交付控制器，可以在 Docker 主机上预配以平衡基于微服务的应用程序的负载。为了提高应用程序交付性能，您需要获得许可的 CPX。Citrix ADC CPX 支持池许可。Citrix ADM 可以充当许可证服务器来许可使用 Citrix ADC CPX 实例。

Citrix 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 Citrix ADM 服务的信息，请参阅 Citrix ADM 服务。

Citrix ADC CPX 许可的类型

Citrix ADC CPX 支持针对本地部署和基于云的部署的带宽和虚拟 CPU（核心）池许可。

带宽池：可以根据实例的带宽消耗情况分配 Citrix ADC CPX 许可证。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，Citrix ADC CPX 仅支持高级带宽池许可。

vCPU 池：在基于 CPU 使用情况的虚拟许可中，许可证指定了特定 Citrix ADC CPX 实例有权使用的 CPU 数量。因此，Citrix ADC CPX 只能从许可证服务器中签出与虚拟 CPU 数量对应的许可证。Citrix ADC CPX 根据系统中运行的 CPU 数量签出许可证。有关 vCPU 池的详细信息，请参阅 Citrix ADC 虚拟 CPU 许可。

Citrix ADC CPX 实例支持的池容量

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
Citrix ADC CPX	40000 注意：这取决于 CPU 频率、生成等。	20 Mbps	1	16	10 Mbps

注意： Citrix 目前正在为基于公有云的产品/服务开发基于 Citrix ADC CPX 消耗量或基于按需付费的许可模式。准备就绪后，它将在公有云市场上提供以供使用。

Citrix ADC CPX 许可的工作原理

Citrix ADC CPX 池容量：一种通用许可证池，Citrix ADC CPX 实例可以从中签出一个实例许可证，并且只能根据需要签出尽可能多的带宽。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。

Citrix ADC CPX 签入和签出许可： Citrix ADM 根据 Citrix ADC CPX 实例的需求分配许可证。在预配 Citrix ADC CPX 实例时，Citrix ADC CPX 实例可以从 Citrix ADM 中签出许可证，并在销毁实例时向 Citrix ADM 重新签回其许可证。

Citrix ADC CPX 行为：单个 Citrix ADC CPX 实例检出高达 1 Gbps 的吞吐量，只能从实例池中签出，而不能从带宽许可证池中签出。Citrix ADC 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 Citrix ADC CPX 实例消耗 1200 Mbps 吞吐量，则将从实例池中使用前 1000 Mbps，剩余的 200 Mbps 将从带宽池中消耗。

Citrix ADC CPX Express

Citrix ADC CPX Express 是一个免费的软件版本，适用于本地部署和云部署。从 Quay 存储库下载 Citrix ADC CPX 实例时，这是不需要许可证文件的 POC 可用的默认容量，它附带以下功能：

- 20 Mbps 带宽

- 最多 250 个 SSL 会话
- 20 Mbps SSL 吞吐量

您必须向 Citrix ADC CPX 实例授权才能升级，以获得更好的性能和生产部署。

Citrix ADC CPX 许可模式

Citrix 为 Citrix ADC CPX 提供了一系列产品许可模式，以满足贵组织的要求。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：

- 来自 ADM 服务的 Citrix ADC CPX 的基于带宽的许可
- 来自 ADM 服务的 Citrix ADC CPX 的基于 vCPU 的许可
- 来自本地 ADM 的 Citrix ADC CPX 的基于带宽的许可
- 来自本地 ADM 的 Citrix ADC CPX 的基于 vCPU 的许可

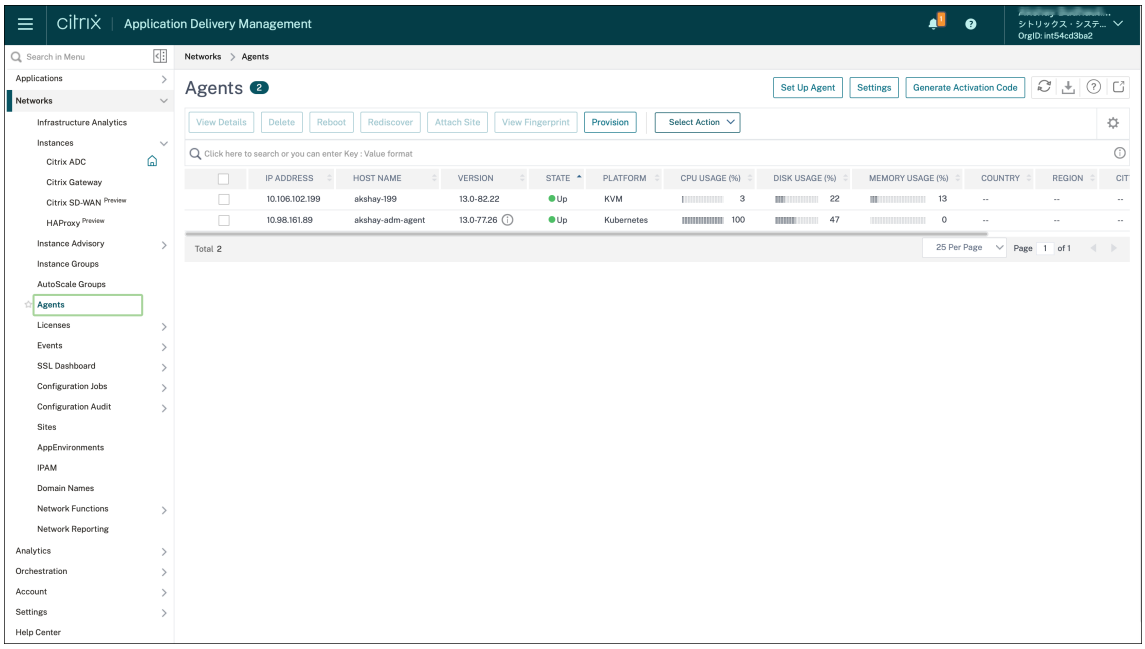
从 Citrix ADM 服务为 Citrix ADC CPX 预配基于带宽和基于 vCPU 的许可

请执行以下步骤，以从 Citrix ADM 服务为 Citrix ADC CPX 预配基于带宽的许可证和基于 vCPU 的许可证。

1. 设置 Citrix ADM。

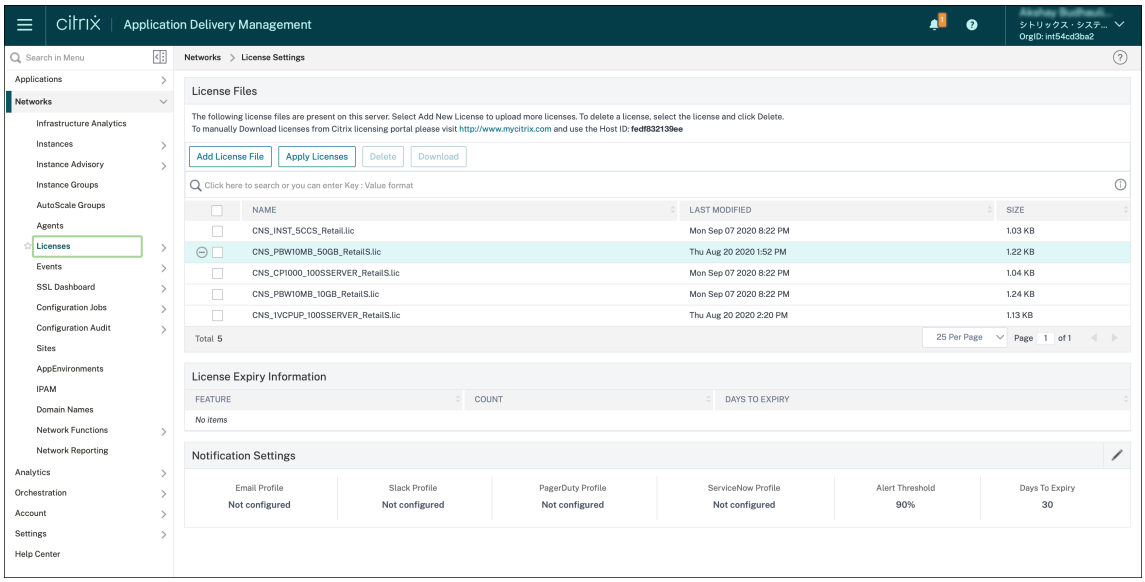
   确保 Citrix ADM 服务设置可以通过 Citrix ADM 代理进行操作。您必须拥有 Citrix ADM 服务和 Citrix ADM 代理帐户，Citrix ADC CPX 许可才能正常使用。有关设置 Citrix ADM 服务和 Citrix ADM 代理的信息，请参阅 Citrix ADM 服务。

   注意：在此过程中，使用虚拟机管理程序（本地）Citrix ADM 代理设置。在下图中，`10.106.102.199` 是用于许可使用 Citrix ADC 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载到 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 Citrix ADC CPX 实例。确保将以下环境变量添加到 Citrix ADC CPX YAML 文件中，以便为 Citrix ADC CPX 实例授权。

   对于来自 Citrix 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：

```
kubectl create namespace bandwidth
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：

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

6. 使用以下命令登录 Citrix ADC CPX 以验证实例化信息：

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

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

```
cli_script.sh “show licenseserver”
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。

从本地 **Citrix ADM** 为 **Citrix ADC CPX** 预配基于带宽的许可和基于 **vCPU** 的许可

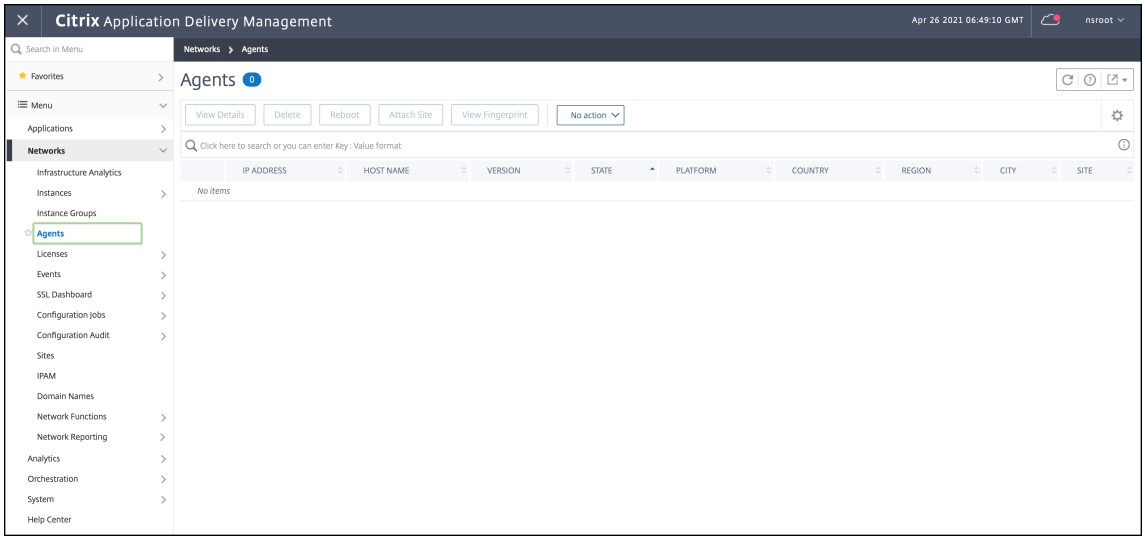
请执行以下步骤，从本地 Citrix ADM 为 Citrix ADC CPX 预配基于带宽和基于 vCPU 的功能。

1. 设置 Citrix ADM。

   确保本地 ADM 设置已准备就绪。确保带或不带用于进行 Citrix ADC CPX 许可的 ADM 代理部署的本地 Citrix ADM 正常运行。

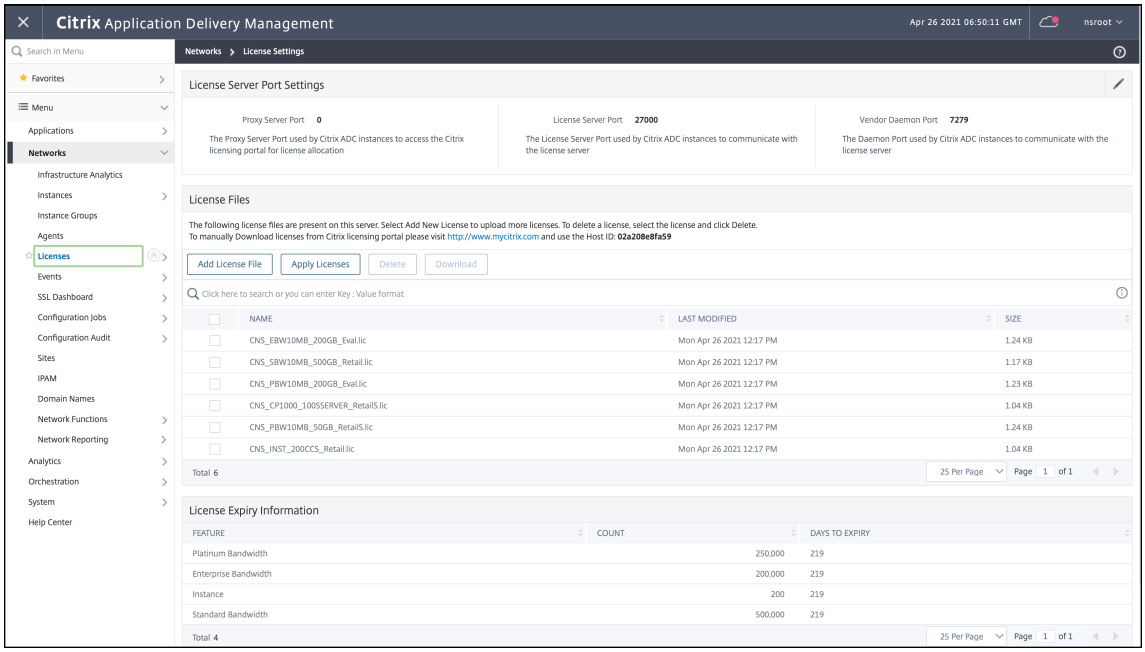
   有关设置本地 Citrix ADM 和 Citrix ADM 代理的信息，请参阅 Citrix ADM 服务。

   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。

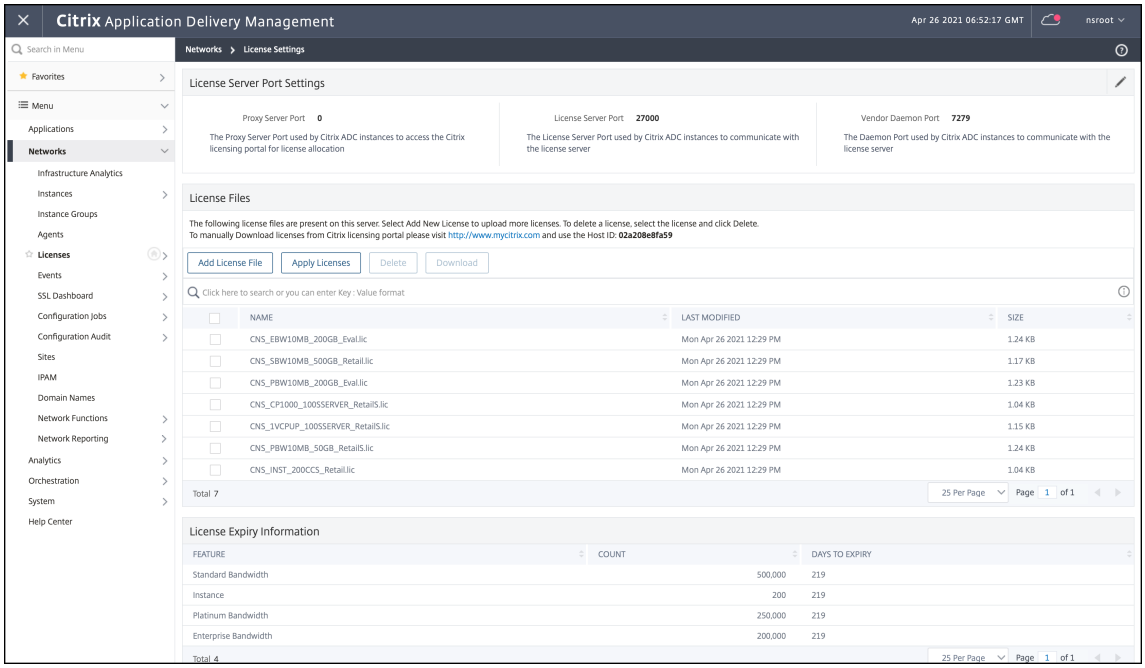


2. 将 Citrix ADC 实例许可证池添加到本地 ADM。

   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载到 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 Citrix ADC CPX 实例。确保将以下环境变量添加到 Citrix ADC CPX YAML 文件中，以便为 Citrix ADC CPX 实例授权。

   对于来自本地 Citrix 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。签出计数等于核心数。

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


```
kubectl create namespace bandwidth
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：

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

6. 使用以下命令登录 Citrix ADC CPX 以验证实例化信息：

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

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

```
cli_script.sh “show licenseserver”
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：

```
kubectl delete -f cpx-bandwidth-license-adm-service.yaml -n bandwidth
kubectl delete -f cpx-core-license-adm-service.yaml -n core
kubectl delete -f cpx-bandwidth-license-adm-onprem.yaml -n bandwidth
kubectl delete -f cpx-core-license-adm-onprem.yaml -n core
kubectl delete namespace bandwidth
kubectl delete namespace core
```

在 **Docker** 中部署 **Citrix ADC CPX** 实例

February 23, 2022

在 Quay 容器注册表中 Citrix ADC CPX 实例以 Docker 映像文件形式提供。要部署实例，请从 Quay 容器注册表下载 Citrix ADC CPX 映像，然后使用 `docker run` 命令或 Docker Compose 工具来部署实例。

必备条件

请确保：

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

    注意：为了获得更加出色的 Citrix ADC CPX 性能，您可以定义希望 Citrix ADC CPX 实例启动的处理引擎数。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：Citrix ADC 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 16.04.5 内核版本 4.4.0-131-generic 上运行 Citrix ADC CPX。

  注意：以下 kubelet 和 kube-proxy 版本存在一些安全漏洞，建议不要在这些版本中使用 Citric ADC 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 **Citrix ADC CPX** 映像

可以使用 `docker pull` 命令从 Quay 容器注册表中下载 Citrix ADC CPX 映像，然后将其部署到您的环境中。使用以下命令可从 Quay 容器注册表中下载 Citrix ADC CPX 映像：

```
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：

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

使用以下命令可验证 Citrix ADC CPX 映像是否安装在 Docker 映像中：

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

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

使用 **docker run** 命令部署 **Citrix ADC CPX** 实例

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

> 重要：
>
> 如果已从 CPX Express 下载 Citrix ADC CPX Express，请务必阅读并理解此处提供的最终用户许可协议 (EULA)：CPX Express，并在部署 Citrix ADC CPX 实例时接受 EULA。

使用以下 ***docker run*** 命令在 Docker 容器中安装 Citrix ADC 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-->
```

此示例基于 Citrix ADC CPX Docker 映像创建一个名为 `mycpx` 的容器。

`-P` 参数是必需的。它告知 Docker 映射 Citrix ADC CPX Docker 映像在容器中公开的端口。这意味着将端口 9080、22、9443 和 161/UDP 映射到从用户定义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 Citrix ADC 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。如果您在部署的主机模式下运行 Citrix ADC CPX，则需要向 Citrix ADC CPX 提供所有系统权限。如果要在桥接模式下使用单个或多个内核运行 Citrix ADC CPX，则可以使用 `--cap-add=NET_ADMIN` 选项来代替此选项。`--cap-add=NET_ADMIN` 选项使您能够以完全网络权限运行 Citrix ADC 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
>
> 如果您要在桥接或无网络中运行 Citrix ADC CPX，则忽略此选项。

`-e NS_NETMODE="HOST"` 是 Citrix ADC CPX 特定的环境变量，允许您指定 Citrix ADC CPX 在主机模式下启动。Citrix ADC CPX 在主机模式下启动后，将在主机上配置 4 个默认 iptables 规则以便对 Citrix ADC CPX 进行管理访问。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
>
> 如果您要在桥接或无网络中运行 Citrix ADC CPX，则忽略此环境变量。

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

注意：Citrix ADC 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 Citrix ADC 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：CPX Express。

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

如果您要在主机网络中运行 Docker ，可以使用 `-e CPX_NW_DEV` 环境变量向 Citrix ADC CPX 容器分配专用网络接口。您需要定义以空格分隔的网络接口。您定义的网络接口由 Citrix ADC CPX 容器保留，直到您卸载 Citrix ADC CPX 容器。预配 Citrix ADC CPX 容器时，会向 Citrix ADC 网络命名空间添加所有分配的网络接口。

> 注意
>
> 如果您在桥接网络中运行 Citrix ADC CPX，则可以更改容器网络，例如配置与容器的另一个网络的连接或者删除现有网络。然后确保重新启动 Citrix ADC 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
```



```
<!--NeedCopy-->
```

-e CPX_CONFIG 是 Citrix ADC CPX 特定的环境变量，通过它可以控制 Citrix ADC CPX 容器的吞吐量性能。Citrix ADC 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 Citrix ADC 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 Citrix ADC CPX 容器在闲置情况下让出 CPU，请定义 { "YIELD" : "Yes" }
- 如果您希望 Citrix ADC 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" }

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

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

-v 参数是可选参数，用于指定 Citrix ADC 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，Citrix ADC 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。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
    CPX_CORES=5 -e CPX_CONFIG='{
 "YIELD":"No" }
 ' -e LS_IP=10.102.38.134 -e PLATFORM=CP1000 --name cpx_host cpx:13.0-x
    .x
<!--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 **Citrix ADC CPX** 的简化版

Citrix 提供了 Citrix ADC CPX 的简化版，该版本消耗的运行时内存较少。Citrix ADC CPX 的简化版可以在服务网格部署中作为 sidecar 进行部署。

Citrix ADC CPX 的简化版支持以下功能：

- 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 Citrix ADC CPX 的简化版，请在执行 `Docker run` 命令时设置 `NS_CPX_LITE` 环境变量。

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

以下示例基于 Citrix ADC CPX 映像创建轻量级容器。

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

默认情况下，在 Citrix ADC CPX 的简化版上禁止使用 `newnslog` 的日志记录。必须在启动 Citrix ADC CPX 的简化版时将 `NS_ENABLE_NEWNSLOG` 环境变量设置为 1，才能将其启用。

以下示例说明了如何在部署 Citrix ADC CPX 的简化版时启用使用 `newnslog` 的日志记录。

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

> 注意： CPX 的简化版仅支持单核 (CPX_CORES=1)。

使用 **Docker Compose** 部署 **Citrix ADC CPX** 实例

可以使用 Docker Compose 工具来预配一个 Citrix ADC CPX 实例或多个 Citrix ADC CPX 实例。必须首先编写一个 Compose 文件，才能使用 Docker Compose 配置 Citrix ADC CPX 实例。此文件指定 Citrix ADC CPX 映像、要为 Citrix ADC CPX 实例打开的端口以及 Citrix ADC CPX 实例的权限。

> 重要
>
> 请确保您已在主机上安装了 Docker Compose 工具。

要预配多个 **Citrix ADC 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 Citrix ADC CPX 映像的存储库和版本。
- **privileged: true** 提供对 Citrix ADC CPX 实例的所有 root 权限。
- **cap_add** 提供对 Citrix ADC CPX 实例的网络权限。
- **<host_directory_path>** 指示 Docker 主机上要为 Citrix ADC CPX 实例装载的目录。
- **<number_processing_engine>** 是您希望 Citrix ADC 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：

```
    <service-name>:
    container_name:
    image: <repository>:<tag>
    ports:
        - 22
        - 9080
        - 9443
        - 161/udp
        - 35021-35030
    tty: true
    cap_add:
        - NET_ADMIN
    ulimits:
        core: -1
    volumes:
        - <host_directory_path>:/cpx
    environment:
        - EULA=yes
        - CPX_CORES=<number_processing_engine>
        - CPX_CONFIG='{
 "YIELD":"Yes" }
 '
<!--NeedCopy-->
```

```
    CPX_0:
    container_name: CPX_0
    image: cpx:13.0-x.x
```



```
    ports:
        -  9443
        -  22
        -  9080
        -  161/udp
        -  35021-35030
    tty: true
    cap_add:
        - NET_ADMIN
    ulimits:
        core: -1
    volumes:
        - /root/test:/cpx
    environment:
        -  CPX_CORES=2
        -  EULA=yes
<!--NeedCopy-->
```

如果要置备一个 Citrix ADC CPX 实例，必须将以下行添加到 compose 文件中：`container_name:<name_of_container>`

运行以下命令预配多个 Citrix ADC 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`

如果要置备一个 Citrix ADC CPX 实例，请运行以下命令：`docker-compose -f <compose_file_name> up -d`

向 **Citrix ADM** 中添加 **Citrix ADC CPX** 实例

February 21, 2022

如果要管理和监视这些实例，必须将安装在 Docker 主机上的 Citrix ADC CPX 实例添加到 Citrix 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。

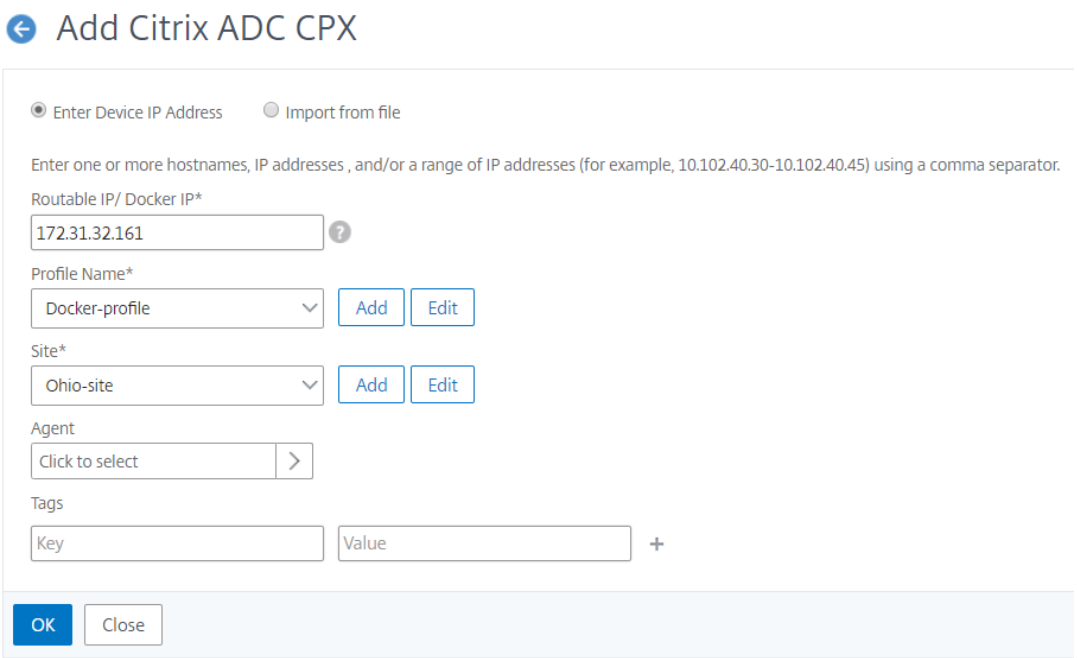
必备条件

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 Citrix ADM 文档。
- 在 Docker 主机上安装了 Citrix ADC CPX 实例。

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

1. 在 Web 浏览器中，键入 **Citrix Application Delivery Management** 的 IP 地址（例如 `http://192.168.100.1`）。
2. 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**。
3. 导航到 **Networks**（网络）**> Instances**（实例）> **Citrix ADC**，然后单击 **CPX** 选项卡。
4. 单击 **Add**（添加）以在 Citrix ADM 中添加新的 CPX 实例。
5. 此时将打开 **Add Citrix ADC CPX**（添加 Citrix ADC CPX）页面。为以下参数输入值：
   a) 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。
   b) 选择 CPX 实例的配置文件。
   c) 选择要在其中部署实例的站点。
   d) 选择代理。
   e) 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。



6. 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）**> Citrix 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 **Citrix ADC CPX** 实例添加到 **Citrix ADM**

还可以使用环境变量将 Citrix ADC CPX 实例添加到 Citrix ADM。要添加实例，必须为 Citrix ADC 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 -（Citrix ADC CPX pod IP 地址不能从外部路由。）
- NS_MGMT_USER - ADM 用户名
- NS_MGMT_PASS - ADM 密码

下面是用于将 Citrix ADC CPX 实例添加到 Citrix ADM 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-->
```

使用 **Kubernetes ConfigMap** 向 **Citrix ADM** 中添加 **Citrix ADC CPX** 实例

Citrix ADC CPX 支持通过 Kubernetes ConfigMap 使用卷装载的文件向 Citrix ADM 注册。要启用此注册方式，Citrix ADC CPX 需要一些环境变量，这些环境变量将与一些通过 ConfigMap 和机密进行卷装载的文件一起指定。

下面是必需的环境变量及其说明：

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

除了列出的环境变量外，Citrix ADC CPX 还需要有关必须向其注册的 ADM 代理的信息。此信息包含 ADM 代理的 IP 地址或 FQDN 详细信息和凭据。Citrix ADC CPX 从卷装载的文件中获取此信息。包含 IP 地址或 FQDN 的 ConfigMap

将作为文件装载到 Citrix ADC CPX 实例的文件系统中。包含 ADM 代理凭据的 Kubernetes 机密也作为文件装载在 Citrix ADC CPX 实例的文件系统中。有了注册所需的所有信息，Citrix ADC CPX 会尝试向 ADM 代理中注册。

下面是通过 ConfigMap 和机密装载为文件的 Citrix ADC CPX YAML 文件片段的示例：

```
        ...
        env:
        - name: "EULA"
          value: "yes"
        - name: "NS_HTTP_PORT"
          value: "9080"
        - name: "NS_HTTPS_PORT"
          value: "9443"
        - name: "NS_SSH_PORT"
          value: "22"
        - name: "NS_SNMP_PORT"
          value: "161"
        - name: "KUBERNETES_TASK_ID"
          value: ""
        ...
        volumeMounts:
          ...
          - mountPath: /var/adminfo/server/
            name: adm-agent-config
          - mountPath: /var/adminfo/credentials/
            name: adm-agent-user
          ...
    volumes:
    ...
    - name: adm-agent-config
      configMap:
        name: adm-agent-config
    - name: adm-agent-user
      secret:
        secretName: adm-secret
```

在上面的示例中，使用了名为 `adm-agent-config` 的 ConfigMap 和机密 `adm-agent-user`。下面是创建所需的 ConfigMap 和机密的示例。

ConfigMap：ConfigMap 是根据名为 `adm_reg_envs` 的文件创建的。该文件需要 ADM 代理的以下格式的 IP 地址或 FQDN：

```
NS_MGMT_SERVER=adm-agent
```

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

请使用以下命令创建 ConfigMap：

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

注意：文件名必须包含 adm_reg_envs 变量，并且必须将其装载到以下路径：/var/adminfo/server/。

机密：请使用以下命令创建 Kubernetes 机密。在以下命令中，user123 为 ADM 代理的用户名，pass123 为密码。

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

即使在群集中部署 ADM 代理之前，也可以在具有所需环境变量和卷装载的文件的 Kubernetes 群集中部署 Citrix ADC CPX 实例。如果您在部署 ADM 代理之前部署 Citrix ADC CPX 实例，Citrix ADC CPX 将继续尝试注册，直到部署 ADM 代理。部署 ADM 代理后，Citrix ADC CPX 实例将使用通过环境变量和卷装载的文件提供的配置数据向 ADM 代理中注册。它可以帮助您避免使用配置信息重新部署 Citrix ADC CPX。

已在 ADM 代理中注册的 Citrix ADC CPX 实例可以在配置更改后动态将注册更改为另一个 ADM 代理。为此，您可以在 ConfigMap 和已部署的 Citrix ADC CPX 的机密中更新配置信息。必须使用新 ADM 代理的 IP 地址或 FQDN 更新从中创建 ConfigMap 的文件，删除旧 ConfigMap，然后创建一个新 ConfigMap。同样，必须删除现有机密，并且必须使用新 ADM 代理的凭据创建新机密。

Citrix ADC CPX License Aggregator

November 9, 2022

Citrix ADC CPX 当前从 Citrix ADM 服务器获取许可证。在 Kubernetes 环境中，Citrix ADC CPX 可以动态启动或关闭。如果 Citrix ADC CPX 意外停机，Citrix ADM 服务器需要几分钟时间才能回收许可证。Citrix ADM 服务器必须能够在 Citrix ADC CPX 出现故障时立即收回此类许可证，这样才能将相同的许可证分配给另一个 Citrix ADC CPX。此外，如果出于任何原因无法访问 Citrix ADM 服务器，则无法在群集中许可使用新的 Citrix ADC CPX。

Citrix ADC CPX License Aggregator 是 Citrix 提供的 Kubernetes 服务。此服务充当 Kubernetes 群集中 Citrix ADC CPX 许可的本地提供程序。在 Kubernetes 群集中部署的 Citrix ADC CPX License Aggregator 服务可以充当 Citrix ADC CPX 与 ADM 许可服务器之间的中介，并跟踪 Citrix ADC CPX 和分配的许可证。借助 Citrix ADC CPX License Aggregator 服务，Citrix ADM 服务器可以在 Citrix ADC CPX 关闭时立即回收许可证。

在 Kubernetes 群集中，Citrix ADC CPX License Aggregator 服务支持作为 sidecar 的 Citrix ADC CPX 和独立部署。

> 注意：
>
> 使用 Citrix ADC CPX 许可证聚合器进行许可需要 Citrix ADC CPX 13.1-30.x 或更高版本。Citrix ADC CPX License Aggregator 不支持许可较旧版本的 Citrix ADC CPX。

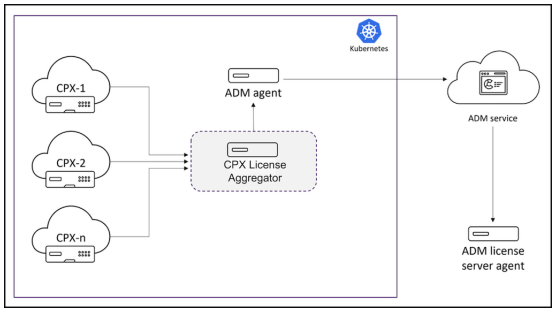
Citrix ADC CPX License Aggregator 的主要优势

下面是使用 Citrix ADC CPX License Aggregator 的主要优势：

- 可扩展性：
  Citrix ADM 许可服务器最多只能支持 10000 个 Citrix ADC CPX 部署。随着 Citrix ADC CPX License Aggregator 服务的推出，每个 Kubernetes 群集都可以充当 Citrix ADM 许可服务器的单个客户端。因此，您可以使用一台 Citrix ADM 许可服务器扩展许多 Citrix ADC CPX。
- 资源优化：
  Citrix ADC CPX License Aggregator 服务还支持群集范围的许可功能，还可以根据需要从 Citrix ADM 服务器签出许可证。Citrix ADC CPX License Aggregator 可以将许可证返还给 Citrix ADM 服务器。
  Citrix ADC CPX License Aggregator 可以处理 Citrix ADC CPX 的不正常终止，并在配置的等待期结束后从此类 Citrix ADC CPX 回收许可证。

部署拓扑

下图显示了 Kubernetes 群集内的 Citrix ADC CPX License Aggregator 部署。



在此图中：

- `CPX` 表示 Citrix ADC CPX
- `CPX License Aggregator` 表示 Citrix ADC CPX License Aggregator

在此示例部署中，Citrix ADC CPX License Aggregator 服务与 Citrix ADC CPX 和 Citrix ADM 代理一起部署在 Kubernetes 群集中。Citrix ADC CPX License Aggregator 服务充当 Citrix ADC CPX 与 Citrix ADM 代理之间的中介，监视群集中的所有 Citrix ADC CPX 并管理其许可证。

使用 Helm 图表部署 Citrix ADC CPX License Aggregator

必备条件

以下必备条件适用：

- 您需要 Kubernetes 版本 1.16 及更高版本。
- 您需要 Helm 3.x 或更高版本。

- 您必须获取具有 Citrix ADC CPX 的许可证的许可证服务器的 IP 地址。
- 必须在 Citrix ADC CPX License Aggregator 中为 Redis 数据库提供密码。您可以使用 Kubernetes 机密提供数据库密码，以下命令可用于创建机密：

```
kubectl create secret generic dbsecret --from-literal=password=<
    custom-password>
```

使用 **Helm** 图表进行部署

根据 Citrix ADC CPX 许可证的类型，执行以下步骤使用 Helm 图表部署 Citrix ADC CPX 许可证聚合器。有关不同类型的 Citrix ADC CPX 许可证的更多信息，请参阅 Citrix ADC CPX 许可。

安装 **Helm** 图表

请使用以下命令添加 Citrix ADC CPX License Aggregator Helm 图表存储库：

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

安装 **Citrix ADC CPX** 许可证聚合器来管理带宽池许可证

根据您拥有的 Citrix ADC CPX 池化许可证的类型，使用以下命令之一。在这些命令中，`my-release` 用作发行版名称。

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

对于 Platinum 带宽许可证：

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

对于 Enterprise 带宽版：

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



```
QUANTUM>,licenseInfo.instanceLowWatermark=<LOW WATERMARK>,
licenseInfo.bandwidthEnterpriseQuantum=<QUANTUM-in-Mbps>,
licenseInfo.bandwidthEnterpriseLowWatermark=<LOW WATERMARK-in-
Mbps>
```

对于 Standard 带宽版：

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

这些命令使用默认配置在 Kubernetes 群集上部署 Citrix ADC CPX 许可证聚合器。可以在安装时配置参数。有关详细信息，请参阅 Helm 图表 GitHub 存储库中的 **Citrix ADC CPX License Aggregator** 配置部分，其中列出了您可以在安装过程中配置的强制性参数和可选参数。

安装 **Citrix ADC CPX** 许可证聚合器来管理 **vCPU** 许可证

根据您拥有的 Citrix ADC CPX vCPU 许可证的类型，使用以下命令之一。在这些命令中，`my-release` 用作发行版名称。

注意：
默认情况下，Helm 图表会安装推荐的 RBAC 角色和角色绑定。

对于 Platinum vCPU 版本：

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

对于 Enterprise vCPU 版本：

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


对于 Standard vCPU 版本：

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

安装 **Citrix ADC CPX** 许可证聚合器来管理多个许可证

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

例如：

要为 `pooled platinum bandwidth edition` 和 `vCPU platinum edition` 许可证部署 Citrix ADC CPX 许可证聚合器，请执行以下操作：

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

将 **Citrix ADC CPX** 配置为从 **Citrix ADC CPX License Aggregator** 获取许可证

当您使用 Citrix ADC CPX License Aggregator 许可 Citrix ADC CPX 时，需要在 Citrix ADC CPX 部署 YAML 中提供环境变量 `CLA`。

必须在此环境变量中提供可用来访问 Citrix ADC CPX License Aggregator 的 `ipaddress` 或 `domainname`，如下所示：

```
env:
- name: "CLA"
  value: "192.0.2.2"
```

或

```
env:
- name: "CLA"
  value: "local-cla.org"
```


您还必须在 Citrix ADC CPX YAML 中提供以下环境变量。

- POD_NAME：指定 pod 的名称。pod 的名称作为环境变量向 Citrix ADC CPX 公开。
- POD_NAMESPACE：指定 pod 的命名空间。pod 的命名空间作为环境变量向 Citrix ADC CPX 公开。
- Bandwidth：指定分配给 Citrix ADC CPX 的带宽（以 Mbps 为单位）。
- Edition：指定许可证版本。支持的值包括 “Standard”、“Platinum” 和 “Enterprise”。
- CPX_CORES：指定要为 Citrix ADC CPX 运行的内核数量。

有关不同 Citrix ADC CPX 许可选项的详细信息，请参阅 Citrix ADC CPX 许可。

下图显示了使用这些环境变量的示例配置：

```
        - name: POD_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
        - name:  “BANDWIDTH”
          value: 1000
        - name:  “CPX_CORES”
          value: 1
        - name:  “EDITION”
          value: PLATINUM
```

还需要在 Citrix ADC CPX YAML 中添加以下标签：

```
      labels:
        adc: citrix
```

有关 Citrix ADC CPX 许可证聚合器的部署示例，请参阅 Citrix ADC CPX 许可证聚合器：部署示例。

配置 **Citrix ADC CPX**

February 23, 2022

可以通过从 Linux Docker 主机访问 CLI 提示窗口或使用 Citrix ADC NITRO API 来配置 Citrix ADC CPX 实例。

使用命令行接口配置 **Citrix ADC 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 Citrix ADC 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 **Citrix ADC CPX** 中使用非默认密码

Citrix ADC CPX 支持为 root 用户帐户使用非默认密码，即 `nsroot`。部署 Citrix ADC 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。
默认的 root 用户帐户密码以纯文本形式存储在 Citrix ADC CPX 文件系统中的 */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 `password` 的 IP 地址，请使用以下命令：

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

使用 **NITRO API** 配置 **Citrix ADC CPX** 实例

可以使用 Citrix ADC NITRO API 配置 Citrix ADC CPX 实例。

要使用 **Nitro API** 来配置 **Citrix ADC 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\
```

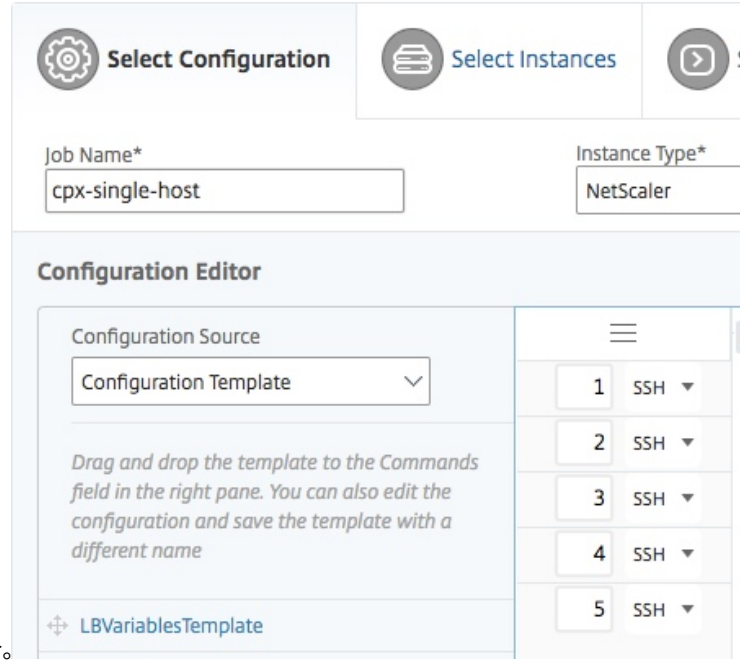
有关使用 NITRO API 的详细信息，请参阅 REST Web 服务。对于 Citrix ADC CPX，请使用 `CPX IP address:port`，其中提及了 `netscaler-ip-address`。

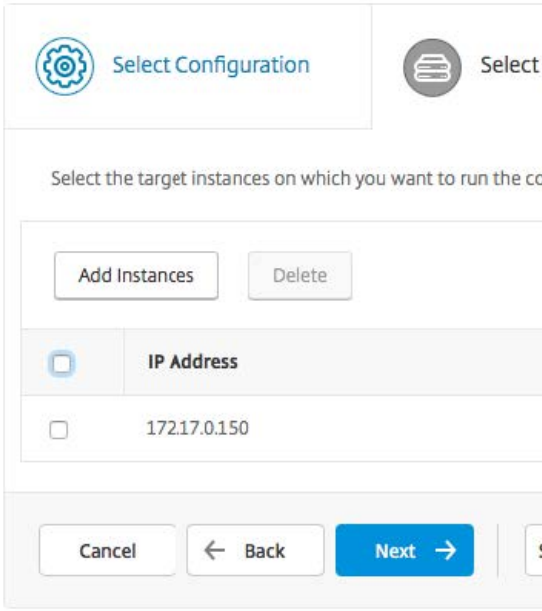
使用作业配置 **Citrix ADC CPX** 实例

可以通过在 Citrix ADM 中创建和运行作业来配置 Citrix ADC 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。之后 Citrix ADM 显示相应的 CLI 命令，让您用于 Citrix ADC CPX 实例。选择配置后，必须选择要对其加载配置的 **Citrix ADC CPX** 实例、指定变量值以及运行作业。

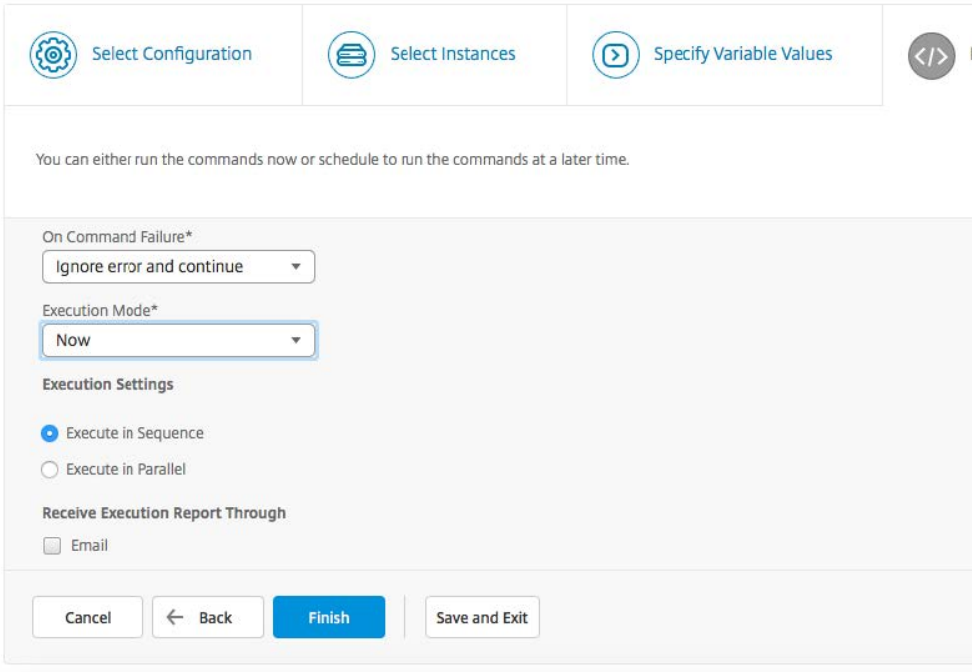
要使用作业配置 **Citrix ADC CPX** 实例，请执行以下操作：

1. 使用管理凭据登录到 Citrix 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。





4. 选择要在其上运行配置的 Citrix ADC CPX 实例，然后单击 **Next**（下一步）。
5. 指定执行设置并单击 “Finish”（完成）以对 Citrix ADC CPX 实例运行命令。如果要保存配置并在以后运行，请单击 **Save and Exit**（保存并退出）。



在 Citrix ADC CPX 实例上配置 AppFlow

December 13, 2021

可以对 Citrix ADC CPX 实例配置 AppFlow 功能，以收集应用程序性能监视和分析所需的 Web 页面性能数据、流和用户会话级别信息以及数据库信息。这些数据记录发送至 Citrix 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。该策略告知 Citrix ADC CPX 选择将其流记录发送到关联操作的请求。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，必须将 Citrix ADC CPX IP 地址指定为导出器的地址。

配置实用程序提供了帮助用户定义策略和操作的工具。它确定 Citrix ADC CPX 如何将特定流的记录导出到一组收集器(操作)。命令行界面为偏好命令行的有经验的用户提供了一组相应的基于 CLI 的命令。

在可以监视记录之前，必须将 Citrix ADC CPX 实例添加到 Citrix ADM。有关将 Citrix ADC CPX 实例添加到 Citrix ADM 的详细信息，请参阅使用 Citrix ADM 安装 Citrix ADC 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：

```
enable ns feature AppFlow
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：

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

要验证配置，请使用以下命令：

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

要使用命令行界面指定多个收集器，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：


```
add appflow collector <collector1> -IPAddress <IP> -Transport Logstream

add appflow collector <collector2> -IPAddress <IP> -Transport Logstream

add appflow action <action> -collectors <collector1> <collector2> -
   Transport Logstream

add appflow policy <policy> true <action> -Transport Logstream

bind lbvserver <lbvserver> -policy <policy> -priority <priority> -
   Transport Logstream
```

配置 **AppFlow** 操作

AppFlow 操作是一个集合收集器，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：

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

要验证配置，请使用以下命令：

```
show appflow action
```

配置 **AppFlow** 策略

配置 AppFlow 操作后，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：

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

show appflow policy <name>
```

绑定 **AppFlow** 策略

要使策略生效，必须将其全局绑定，以便其适用于流经 Citrix ADC 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：


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

使用以下命令验证配置：

```
show appflow global
```

使用配置文件配置 **Citrix ADC CPX**

September 7, 2021

可以在部署 Citrix ADC CPX 实例过程中使用静态配置文件配置 Citrix ADC CPX，代替使用命令行接口 (cli_script.sh)、NITRO API 或 Citrix ADM 配置作业来配置 Citrix ADC CPX。

可以在部署 Citrix ADC CPX 容器过程中提供一个静态配置文件作为输入文件。Citrix ADC 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。此配置中包含 Citrix ADC 特定的配置以及要在 Citrix ADC CPX 容器上动态运行的 bash shell 命令。

静态配置文件的结构

如之前所述，部署了 Citrix ADC CPX 时，将根据在静态配置文件中指定的配置对其进行配置。

静态配置文件是一个包括两个标记 ##NetScaler Commands 和 ##Shell Commands 的 .conf 文件。在标记 ##NetScaler Commands 下，必须添加所有 Citrix ADC 命令以在 Citrix ADC CPX 上配置 Citrix ADC 特定的配置。在标记 ##Shell Commands 下，必须添加要在 Citrix ADC CPX 上运行的 shell 命令。

Citrix ADC CPX 容器部署过程中，将按在配置文件中指定的顺序在容器上运行 Citrix ADC 命令和 shell 命令。

> 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 Citrix ADC 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。
> - 重新启动 Citrix ADC CPX 容器时，将在容器上重新应用配置文件。

```
#NetScaler Commands

add lb vserver v1 http 1.1.1.1 80

add service s1 2.2.2.2 http 80

```



```
bind lb vserver v1 s1

#Shell Commands

touch /etc/a.txt

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

#NetScaler Commands

add lb vserver v2 http

#Shell Commands

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

#NetScaler Commands

add lb vserver v3 http

#This is a test configuration file
<!--NeedCopy-->
```

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

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

Citrix ADC CPX 中的动态路由支持

December 13, 2021

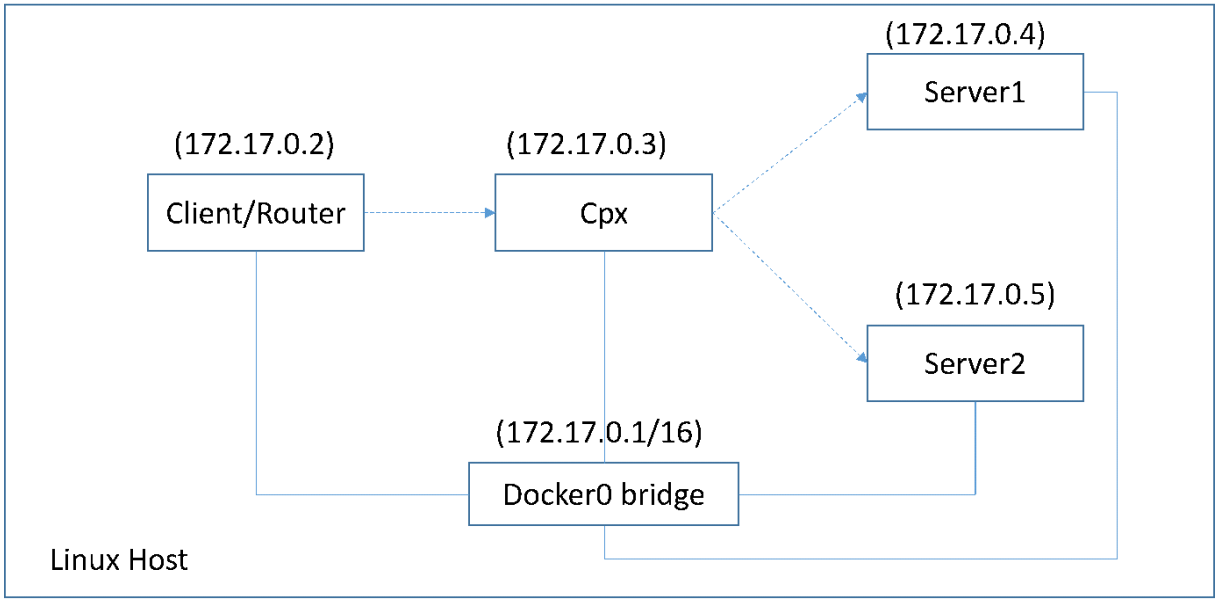
Citrix ADC CPX 支持 BGP 动态路由协议。动态路由协议的关键目标是根据绑定到虚拟服务器的服务的运行状况公告虚拟服务器的 IP 地址。它有助于上游路由器在通往按地形分布的虚拟服务器的多条路由中选择最佳路由。

有关 Citrix ADC CPX 中的非默认密码的信息，请参阅配置 Citrix ADC CPX 文档中的`Support for using a non-default password in Citrix ADC CPX`部分。

在单主机网络中，客户端、服务器和 Citrix ADC 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，Citrix ADC 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 Citrix ADC 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。

执行以下步骤以在具有桥接联网模式的单个 Docker 主机中的 Citrix ADC CPX 上配置 BGP。

在 **Citrix ADC 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 Citrix ADC CPX 映像创建容器：

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

例如：

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

2. 使用以下命令登录容器：

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

3. 使用以下命令启用 BGP 功能：

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

4. 使用 `show ns ip` 命令获取 NSIP：


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

5. 使用以下命令添加虚拟服务器：

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

6. 添加服务并将服务绑定到虚拟服务器。

7. 使用以下命令为 VIP 启用 hostroute：

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

从容器注销并将 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：

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

按如下所示指定命令：

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

9. 使用以下 NITRO 命令将获取的 BGP 路由安装到 PE 中：

```
curl -u username:password http://<NSIP>:9080/nitro/v1/config/ --
   data 'object={
 "params":{
 "action":"apply" }
 ,"routerDynamicRouting": {
```



```
"commandstring" : "ns route-install bgp" }
 }
'
```

10. 使用以下 NITRO 命令验证 BGP 邻近状态：

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

示例输出：

```
root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/
   nitro/v1/config/routerDynamicRouting/bgpRouter
{
 "errorcode": 0, "message": "Done", "severity": "NONE", "
    routerDynamicRouting":{
"bgpRouter":[{
 "localAS": 100, "routerId": "172.17.0.3", "afParams": [ {
 "addressFamily": "ipv4"  }
, {
 "addressFamily": "ipv6"  }
 ], "neighbor": [ {
 "address": "172.17.0.2", "remoteAS": 101, "ASOriginationInterval
    ": 15, "advertisementInterval": 30, "holdTimer": 90, "
    keepaliveTimer": 30, "state": "Connect", "singlehopBfd":
    false, "multihopBfd": false, "afParams": [ {
 "addressFamily": "ipv4", "activate": true  }
, {
 "addressFamily": "ipv6", "activate": false  }
 ]
```

11. 使用以下命令验证通过 BGP 获取的路由是否安装在数据包引擎中：

```
cli_script.sh  “show route”
```

12. 使用以下命令保存配置：

```
cli_script.sh  “save config”
```

动态路由配置保存在 /nsconfig/ZebOS.conf 文件中。

为 **Citrix ADC CPX** 配置高可用性

December 13, 2021

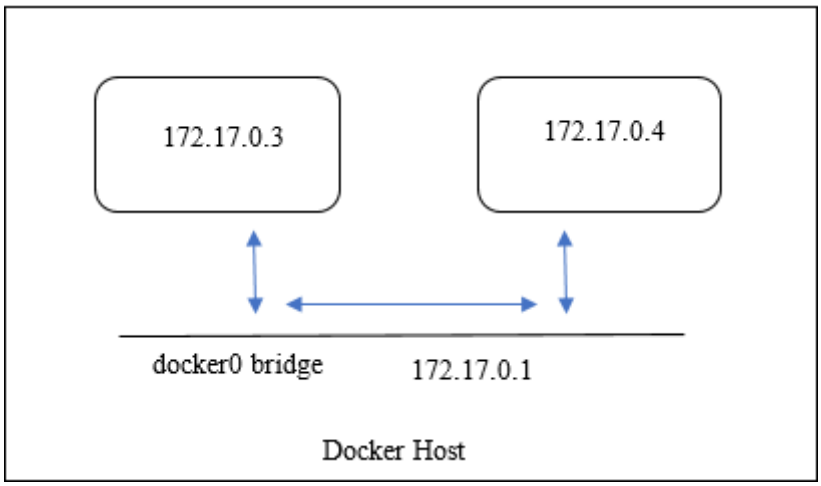
具有任务关键型和业务关键型应用程序的系统必须持续可用，而不会出现单点故障。具有高可用性的系统可确保应用程序的持续可用性，而不会中断为用户提供的服务。Citrix ADC CPX 支持两个 Citrix ADC 实例的高可用性部署，以保护服务免受计划外停机的影响，并确保出现故障时的业务连续性。配置高可用性后，还可以升级 Citrix ADC CPX 软件，而不会造成用户的服务中断。

> 注意：
> 如果禁用了内部用户帐户，则不支持 Citrix ADC CPX 功能的高可用性。

拓扑 **1**：在具有桥接联网模式的单个 **Docker** 主机上部署 **Citrix ADC CPX** 实例

在此拓扑中，两个 Citrix ADC CPX 节点将在具有桥接联网模式的同一个 Docker 主机上创建。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，在同一个 Docker 主机上创建了两个 Citrix ADC CPX 实例 CPX-1（NSIP：172.17.0.3）和 CPX-2 (NSIP：172.17.0.4)。要获得高可用性支持，必须使用另一个节点的 NSIP 在两个 Citrix ADC CPX 实例上配置高可用性节点。

执行以下步骤以在桥接模式下的单个 Docker 主机上的 Citrix ADC CPX 实例上配置高可用性支持。

1. 访问 Docker 主机并登录 Citrix ADC CPX 实例的 SSH 提示窗口，如下图所示。有关详细信息，请参阅使用命令行接口配置 Citrix ADC 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。


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

> 注意：
> 当重新启动处于桥接联网模式的 Citrix ADC CPX 节点时，分配给 Citrix ADC CPX 的 IP 地址可能会根据主机上的 Docker 版本而变化。如果两个节点中的任一节点的 NSIP 在重新启动 Citrix ADC CPX 后发生变化，则即使已保存配置，现有的高可用性配置也无法正常运行。在这种情况下，您必须重新在 Citrix ADC CPX 节点上配置高可用性。

拓扑 2：使用桥接联网模式在不同的 Docker 主机上部署 Citrix ADC CPX

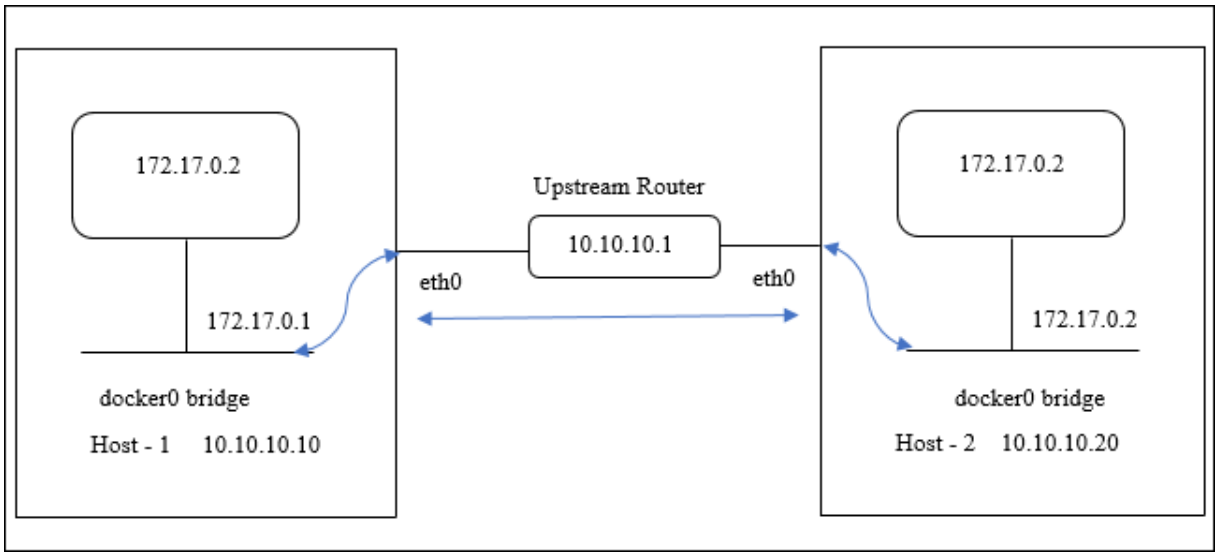
在此拓扑中，两个 Citrix ADC CPX 实例以桥接模式部署在两个不同的 Docker 主机上，这些主机可以互相访问。在此部署中，Citrix ADC CPX 必须知道主机的 IP 地址。可以在预配 Citrix ADC CPX 时使用 **HOST** 环境变量，以使 Citrix ADC CPX 知晓主机的 IP 地址。

必须为 Citrix ADC CPX 节点设置端口映射。可以在创建 Citrix ADC CPX 节点时使用 **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

下图解释了在两个不同的 Docker 主机上以桥接模式部署两个 Citrix ADC CPX 实例的拓扑结构。



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，只有一个 Citrix ADC CPX 可以构成高可用性对。同一主机上的任何其他 Citrix ADC CPX 都不能与另一台主机上的另一个 Citrix ADC CPX 构成高可用性对。

执行以下步骤以桥接模式在不同的 Docker 主机上部署 Citrix 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 上使用所需的端口映射部署 Citrix ADC CPX。

```
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. 使用与主机 2 的 IP 地址相同的命令在 host2 上部署 Citrix ADC CPX。

```
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。

```
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。

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

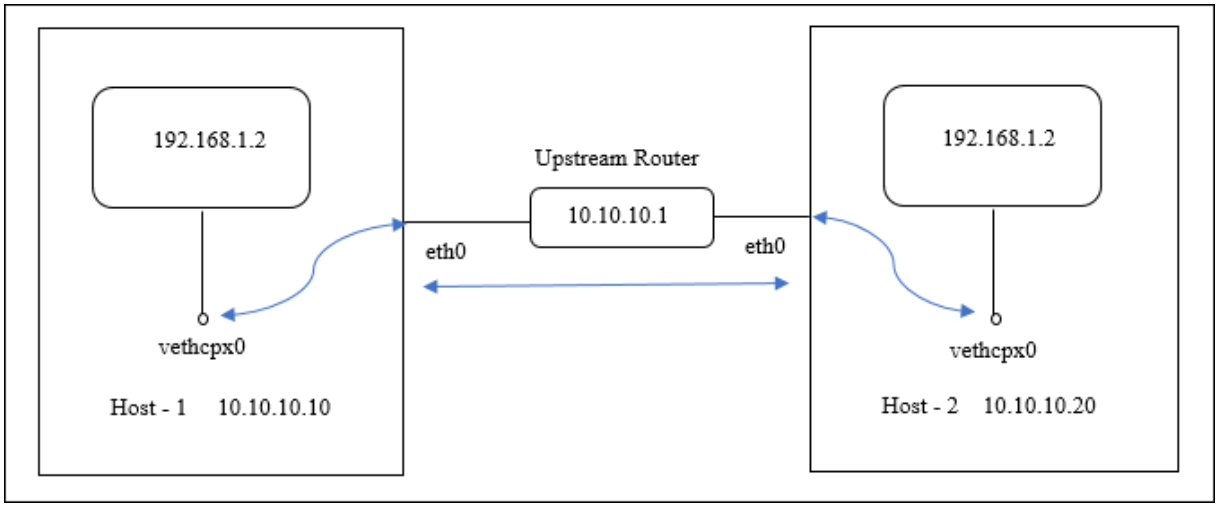
> 注意：在此部署中，必须使用高可用性节点的主机 IP 地址，而非高可用性节点的 NSIP 地址。

拓扑 **3**：在没有专用接口的情况下以主机联网模式在不同的 **Docker** 主机上部署 **Citrix ADC CPX**

在此拓扑中，两个 Citrix ADC CPX 实例在主机模式下部署在两个不同的 Docker 主机上，而无需专用接口。主机必须可以相互访问。
在此部署中，Citrix ADC CPX 必须知道主机的 IP 地址。可以在预配 Citrix ADC CPX 过程中使用 **HOST** 环境变量来使其知晓主机的 IP 地址。

必须为 Citrix ADC CPX 节点设置端口映射。可以在创建 Citrix ADC CPX 节点时使用 **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 Citrix ADC CPX。

请执行以下步骤以部署 Citrix ADC 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 上使用所需的端口映射部署 Citrix ADC CPX。

```
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. 使用以下命令通过 host2 的 IP 地址在 host2 上部署 Citrix ADC CPX。

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1
--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。

```
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。

```
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，以主机联网模式将两个 Citrix ADC CPX 实例部署在不同的 Docker 主机上。主机必须有多个接口。可以使用 **CPX_NW_DEV** 环境变量为 Citrix ADC CPX 指定专用接口。

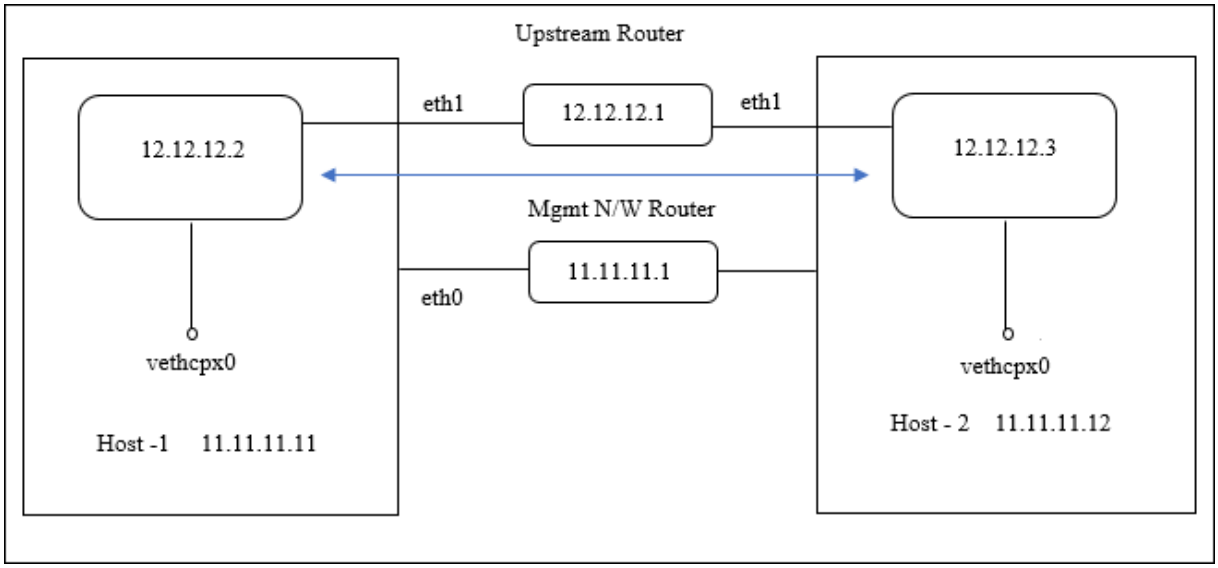
有关使用 CPX_NW_DEV 环境变量为 Citrix ADC CPX 分配专用网络接口的详细信息，请参阅使用 Docker 运行命令部署 Citrix ADC CPX 实例。

部署在不同 Docker 主机上的 Citrix ADC CPX 必须可以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 Citrix ADC 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，两个主机模式 Citrix ADC CPX 在两台不同的主机上创建。Citrix ADC CPX 实例在两台主机上拥有 **eth1** 接口，而 **eth1** 接口连接到同一网络。

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 Citrix ADC CPX。

请执行以下步骤以部署 Citrix ADC 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 Citrix ADC CPX。

```
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 host2 上以主机模式部署 Citrix ADC CPX。

```
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 Citrix ADC CPX 节点配置静态路由，才能到达另一个 Citrix ADC CPX 节点以交换检测信号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。

```
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。

```
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

December 13, 2021

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 Citrix ADC 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，请参阅配置日志记录驱动程序。

默认情况下，Citrix ADC 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 Citrix ADC 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，Citrix ADC 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。

```
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
<!--NeedCopy-->
```

同样，在下例中，Citrix ADC 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。

```
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
<!--NeedCopy-->
```

升级 **Citrix ADC CPX** 实例

December 13, 2021

可以通过关闭 Citrix ADC CPX 实例、在同一装载点上安装最新版本的 Citrix ADC CPX，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 Citrix ADC CPX 实例的 **/cpx** 目录，装载点为 **/var/cpx**，Citrix ADC CPX 装载目录为 **/cpx**，如下所示：

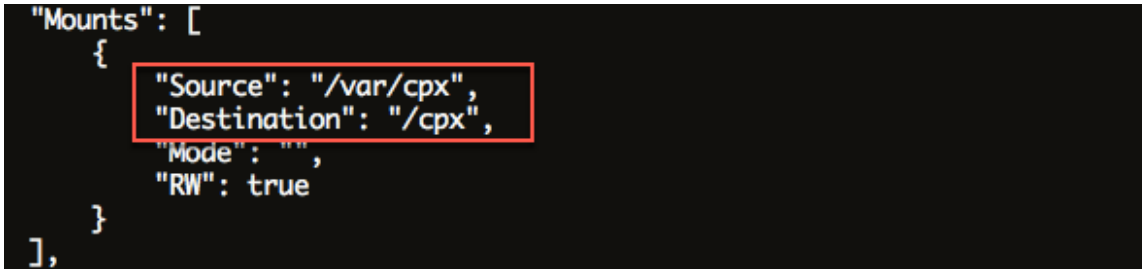

```
    root@ubuntu:~# docker run -dt -e EULA=yes --name mycpx -v /var/cpx
       :/cpx --ulimit core=-1 cpx:13.0-x.x
<!--NeedCopy-->
```

必备条件

请确保您具有：

- 具有您装载了现有 Citrix ADC 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 Citrix ADC 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。



- 下载最新的 Citrix ADC CPX Docker 映像文件并加载 Citrix ADC 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 Citrix ADC CPX 映像后，可以输入 docker images 命令以显示与映像有关的信息：

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

    root@ubuntu:~# docker images

    REPOSITORY   TAG         IMAGE ID         CREATED     VIRTUAL SIZE

    cpx          13.0-x.x  2e97aadf918b  43 hours ago      414.5 MB
<!--NeedCopy-->
```

升级 **Citrix ADC CPX** 实例

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

```
root@ubuntu:~# docker stop mycpx
mycpx
<!--NeedCopy-->
```


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

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

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

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

    CONTAINER ID   IMAGE   COMMAND
       CREATED   STATUS   PORTS
                          NAMES

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

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

```
root@ubuntu:~# docker rm mycpx
mycpx
<!--NeedCopy-->
```

在 **Citrix ADC CPX** 实例中使用通配符虚拟服务器

September 7, 2021

预配 Citrix ADC 实例时，Docker 引擎只为一个 Citrix ADC CPX 实例分配一个专用 IP 地址（单一 IP 地址）。Citrix 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 地址来添加负载平衡 (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，Citrix ADC CPX 将替换通配符虚拟服务器的已分配 IP 地址。已分配的 IP 地址为 127.0.0.1，该地址替换为 Docker 引擎分配给 Citrix ADC CPX 实例的 NSIP。

在高可用性 Citrix ADC CPX 部署中，您可以在主 Citrix ADC CPX 实例上添加通配符虚拟服务器。节点之间的配置同步用于在辅助 Citrix ADC CPX 实例上配置通配符虚拟服务器。它不需要在 Docker 引擎分配给 Citrix ADC 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。

要创建通配符负载平衡虚拟服务器，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：

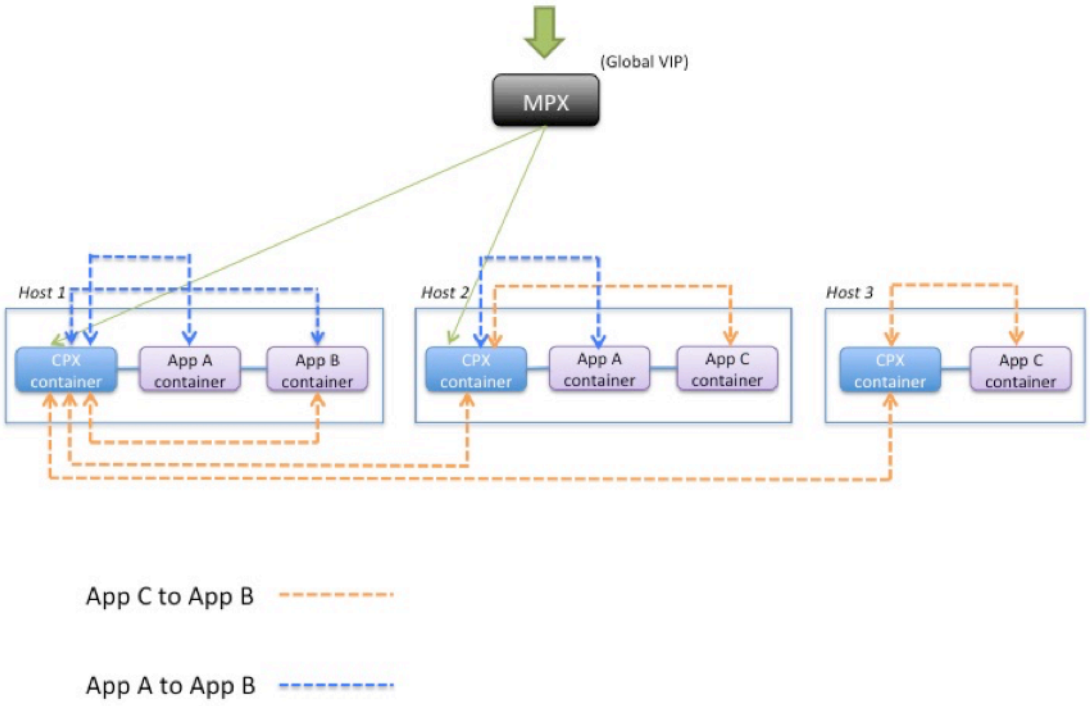
```
1      add cs vserver <name> <serviceType> 127.0.0.1 <port>
2
3      add cs vserver testcsvserver HTTP 127.0.0.1 30000
4  <!--NeedCopy-->
```

将 **Citrix ADC CPX** 部署为代理以支持东西通信流

September 7, 2021

在此部署中，Citrix ADC 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。Citrix ADC 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。

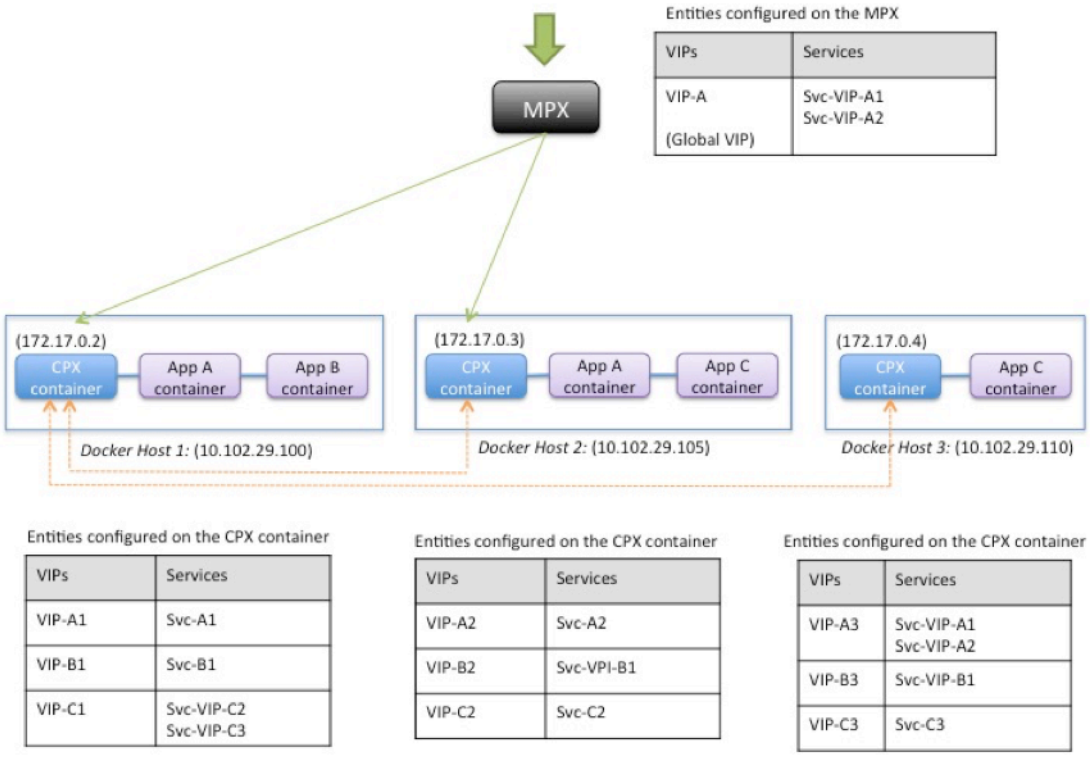
下图说明了两个应用程序之间通过 Citrix ADC CPX 实例进行的通信流。



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 Citrix ADC CPX 容器先接收该请求。然后 Citrix ADC CPX 容器将流量传送至与应用程序 B 在同一主机上托管的 Citrix ADC 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，Citrix ADC 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 Citrix ADC MPX 的流量在 Citrix ADC 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 Citrix ADC 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 172.17.02:30002	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				

为了配置此示例方案，请在所有三个 Docker 主机上创建 Citrix ADC 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=truecpx:6.2
2 <!--NeedCopy-->
```

使用 Citrix 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 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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 **Citrix ADC CPX**

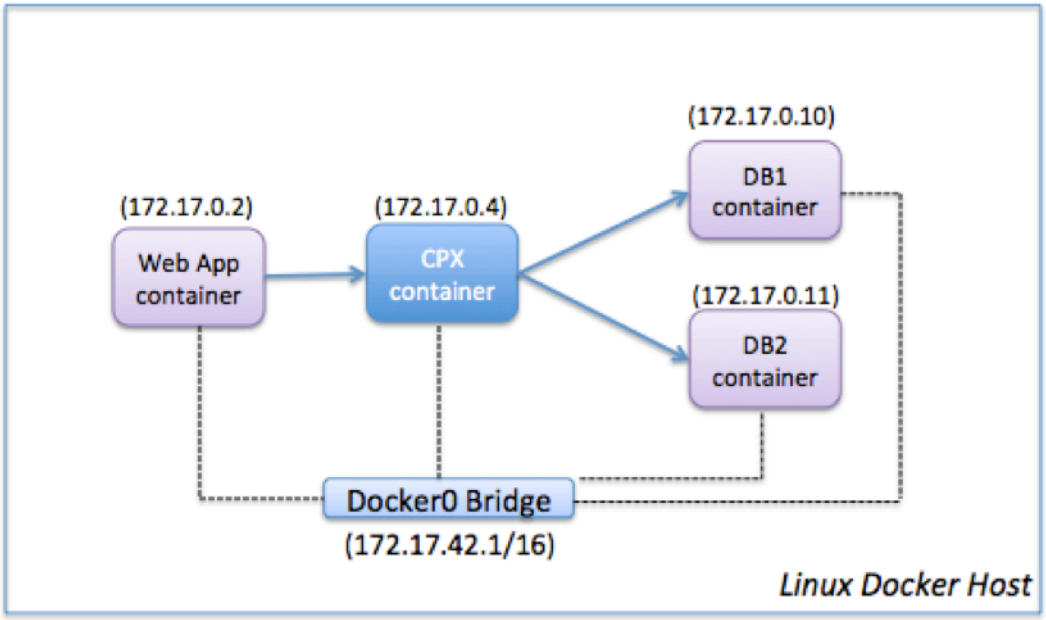
September 7, 2021

在单主机网络中，Citrix ADC 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，Citrix ADC 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 Citrix ADC 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，Citrix ADC 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。Citrix ADC 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 Citrix ADC CPX 与数据库容器通信，必须先在 Citrix ADC CPX 容器上配置两个服务以代表两个服务器。然后使用 Citrix ADC CPX IP 地址和非标准 HTTP 端口（例如 81）（Citrix ADC 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：


```
    add service db1 HTTP 172.17.0.10 80
    add service db2 HTTP 172.17.0.11 80
    add lb vserver cpx-vip HTTP 172.17.0.4 81
    bind lb vserver cpx-vip db1
    bind lb vserver cpx-vip db2
<!--NeedCopy-->
```

在多主机网络中部署 **Citrix ADC CPX**

September 7, 2021

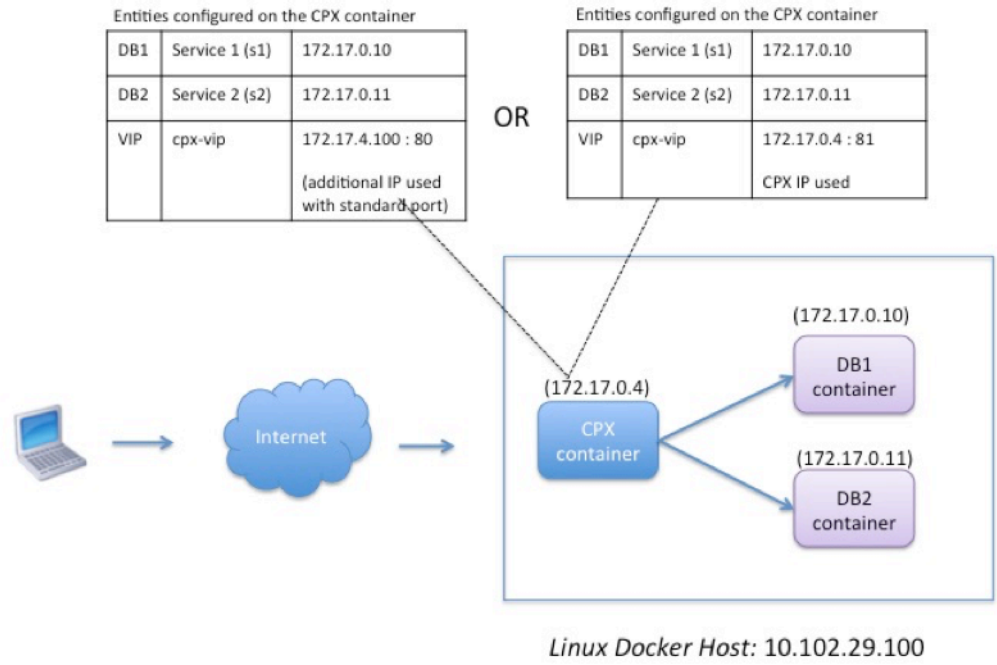
可以在数据中心的生产部署中配置多主机网络中的 Citrix ADC CPX 实例，在那里它提供负载平衡功能。此外它还提供监视功能和分析数据。

在多主机网络中，Citrix ADC 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，在这些部署中，Citrix ADC CPX 实例对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**：**Citrix ADC 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 主机上预配 Citrix ADC 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。此拓扑可在生产部署中使用，在该部署中，Citrix ADC CPX 实例对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 主机上预配 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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。

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

方法 **2**：将 Citrix ADC CPX IP 地址用于 VIP，并配置端口映射：

1. 在 Citrix ADC 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。

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

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

<!--NeedCopy-->
```

预配 Citrix ADC CPX 实例后，使用 Citrix ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```
    add service s1 172.17.0.10 http 80
    add service s2 172.17.0.11 http 80
    add lb vserver cpx-vip HTTP 172.17.0.4 81
    bind lb vserver cpx-vip s1
    bind lb vserver cpx-vip s2
<!--NeedCopy-->
```

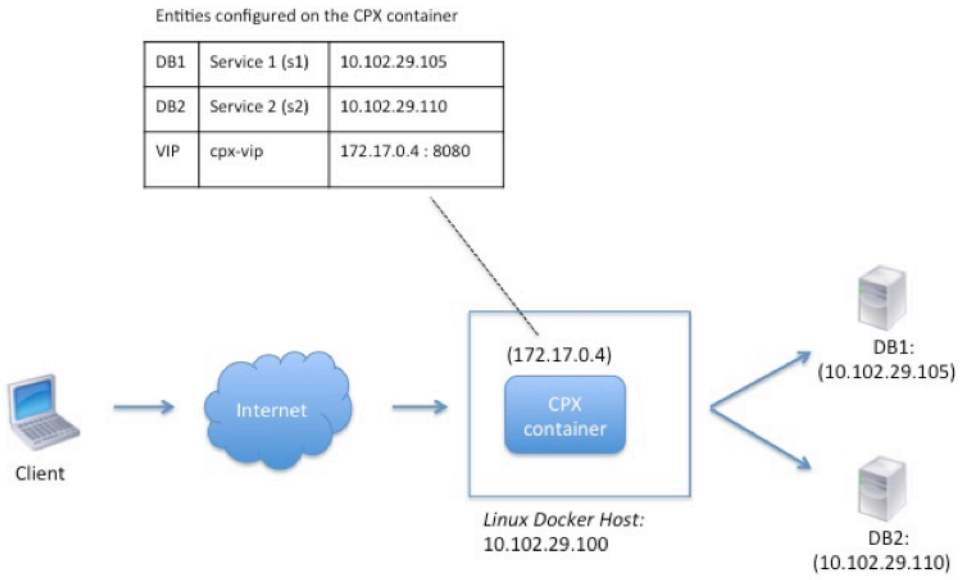
> 注意：
>
> 如果在预配 Citrix ADC CPX 实例过程中未配置端口映射，请通过在 Linux shell 提示窗口中运行以下命令来配置 NAT 规则：
>
> **iptables -t nat -A PREROUTING -p tcp -m addrtype –dst-type LOCAL -m tcp –dport** 50000 **-j DNAT –to-destination** 172.17.0.4:81

拓扑 **2**：具有物理服务器和客户端的 **Citrix ADC CPX**

在此拓扑中，在 Docker 主机上只预配了 Citrix ADC CPX 实例。客户端和服务器不是基于容器的，且位于网络的其他地方。

在此环境中，您可以配置 Citrix ADC 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，Citrix ADC 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 Citrix ADC 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 Citrix ADC CPX 进行通信，必须先在创建 Citrix ADC CPX 容器时配置端口映射。然后在 Citrix ADC CPX 容器上配置两个服务以代表两台服务器。最后，使用 Citrix ADC 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。

为了配置此示例方案，请在创建 Citrix ADC 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-->
```

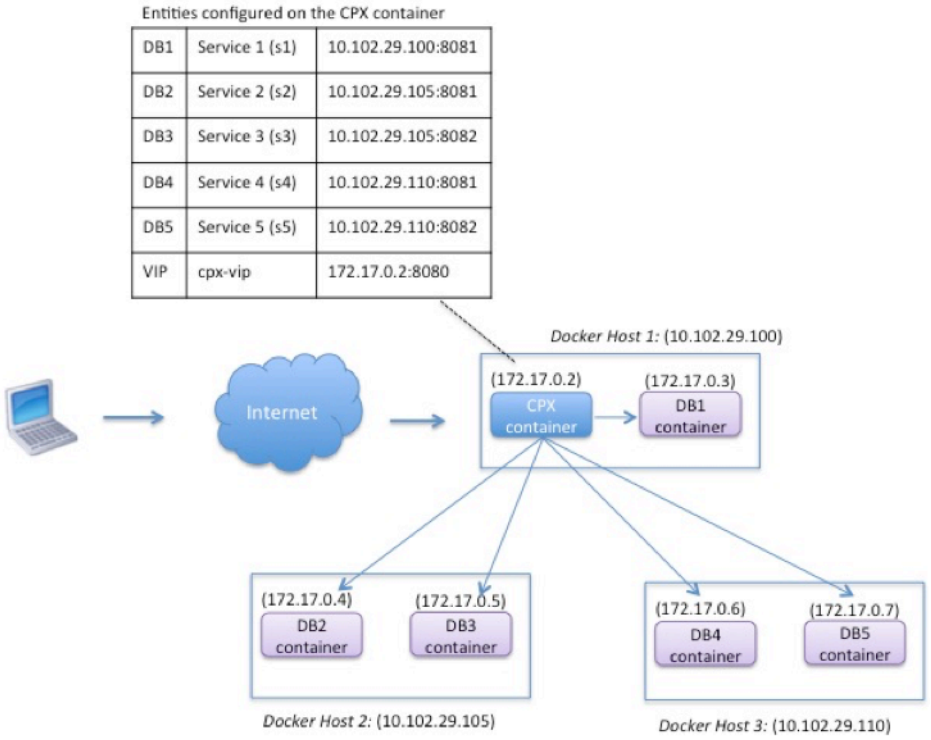
然后，使用 Citrix ADM 中的作业功能或使用 NITRO API 来运行以下命令：

```
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**：在不同的主机上预配 **Citrix ADC CPX** 和服务器

在此拓扑中，在不同的 Docker 主机上预配 Citrix ADC CPX 实例和数据库服务器，客户端流量源自 Internet。此拓扑可在生产部署中使用，在该部署中，Citrix ADC CPX 实例对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 主机上预配了 Citrix ADC CPX 实例和服务器 (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 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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 Citrix ADC 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。

为了配置此示例方案，请在创建 Citrix ADC 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 Citrix 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 **Citrix ADC CPX** 部署对网络的直接访问

September 7, 2021

在桥接网络连接模式下，可以配置 Citrix ADC CPX 实例以便直接访问网络。在此方案中，传入流量直接在 Citrix ADC 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平衡，然后将网络公用 IP 地址配置为 Citrix ADC 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;
```



```
    ip route del default;
    ip route add default via 10.102.29.1 dev docker0
<!--NeedCopy-->
```

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

```
    add service s1 172.17.0.8 http 80
    add service s2 172.17.0.9 http 80
    add lb vserver cpx-vip HTTP 10.102.29.102 80
    bind lb vserver cpx-vip s1
    bind lb vserver cpx-vip s2
<!--NeedCopy-->
```

使用 **ConfigMap** 在 **Kubernetes** 中配置 **Citrix ADC CPX**

September 7, 2021

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

创建一个包含 Citrix ADC 特定的配置以及要在 Citrix ADC CPX 实例上动态运行的 bash shell 命令的 `cpx.conf` 配置文件。配置文件结构需要两种类型的标记，即 `##Citrix ADC Commands` 和 `##Shell Commands`。在标记 `##Citrix ADC Commands` 下，必须添加所有 Citrix ADC 命令以在 Citrix ADC CPX 实例上配置 Citrix ADC 特定的配置。在标记 `##Shell Commands` 下，必须添加要在 Citrix ADC 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 Citrix ADC CPX 容器时存在故障情形，故障将记录在 `ns.log` 文件中。
> - Citrix ADC CPX 实例启动后，如果更改了 ConfigMap，则仅当重新启动 Citrix ADC CPX 实例时才能应用更新后的配置。

下面是示例配置文件：

```
#Citrix ADC Commands
add lb vserver v1 http 1.1.1.1 80
add service s1 2.2.2.2 http 80
bind lb vserver v1 s1
#Shell Commands
touch /etc/a.txt
echo "this is a" > /etc/a.txt
```



```
#Citrix ADC Commands
add lb vserver v2 http
#Shell Commands
echo "this is a 1" >> /etc/a.txt
#Citrix ADC Commands
add lb vserver v3 http
<!--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。

```
kubectl create configmap cpx-config --from-file=cpx.conf
<!--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 Citrix ADC 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：

```
    apiVersion: v1
    data:
        cpx.conf: |
        #Citrix ADC Commands
            add lb vserver v1 http 1.1.1.1 80
            add service s1 2.2.2.2 http 80
            bind lb vserver v1 s1
        #Shell Commands
            touch /etc/a.txt
            echo "this is a" > /etc/a.txt
            echo "this is the file" >> /etc/a.txt
            ls >> /etc/a.txt
        #Citrix ADC Commands
            add lb vserver v2 http
        #Shell Commands
            echo "this is a 1" >> /etc/a.txt
        #Citrix ADC Commands
            add lb vserver v3 http
        #end of file
    kind: ConfigMap
    metadata:
        creationTimestamp: 2017-12-26T06:26:50Z
        name: cpx-config
        namespace: default
        resourceVersion: "8865149"
```



```
        selfLink: /api/v1/namespaces/default/configmaps/cpx-config
        uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
<!--NeedCopy-->
```

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

```
apiVersion: v1
kind: Pod
metadata:
  name: cpx-1
  labels:
    app: cpx-daemon
  annotations:
    NETSCALER_AS_APP: "True"
spec:
  hostNetwork: true
  containers:
  - name: cpx
    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-36.28"
    securityContext:
      privileged: true
    volumeMounts:
    - name: config-volume
      mountPath: /cpx/conf
    env:
      - name: "EULA"
        value: "yes"
      - name: "NS_NETMODE"
        value: "HOST"
      - name: "kubernetes_url"
        value: "https://10.90.248.101:6443"
      - name: "NS_MGMT_SERVER"
        value: "10.90.248.99"
      - name: "NS_MGMT_FINGER_PRINT"
        value: "19:71:A3:36:85:0A:2B:62:24:65:0F:7E:72:CC:DC:AD:B8:BF
            :53:1E"
      - name: "NS_ROUTABLE"
        value: "FALSE"
      - name: "KUBERNETES_TASK_ID"
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
    imagePullPolicy: Never
  volumes:
  - name: config-volume
```



```
39      configMap:
40        name: cpx-config
41  <!--NeedCopy-->
```

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

将 **Citrix ADC CPX** 部署为 **Kubernetes** 节点的本地 **DNS** 缓存

February 21, 2022

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，您可以将 Citrix ADC CPX 部署为每个 Kubernetes 节点上的本地 DNS 缓存，并将来自节点中的应用程序 pod 的 DNS 请求转发到 Citrix ADC 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 Citrix ADC CPX，Kubernetes DaemonSet 实体用于在 Kubernetes 群集中的每个节点上安排 Citrix ADC CPX pod。Kubernetes DaemonSet 可确保群集中的每个 Kubernetes 节点上都有 Citrix ADC CPX 的实例。
要使应用程序 pod 将流量定向到 CPX DNS pod，您需要创建一个 Kubernetes 服务，该服务使用端点作为 Citrix ADC CPX pod。此服务的群集 IP 用作应用程序 pod 的 DNS 端点。要确保应用程序 pod 使用 Citrix ADC CPX 服务群集 IP 地址进行 DNS 解析，您需要更新每个使用 Citrix ADC CPX 服务群集 IP 的节点上的 kubelet 配置文件。

引入了以下环境变量来支持将 Citrix ADC CPX 部署作为 NodeLocal DNS 缓存：

- `KUBE_DNS_SVC_IP`：指定 `kube-dns` 服务的群集 IP 地址，这是在 Citrix ADC CPX pod 上触发配置的必填参数。当 Citrix ADC CPX 缓存中没有 DNS 查询响应时，Citrix ADC CPX pod 会将 DNS 查询定向到此 IP 地址。
- `CPX_DNS_SVC_IP`：指定 Citrix ADC 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，将添加一条 `iptables` 规则，以将来自应用程序 pod 的 DNS 请求定向到节点内的本地 Citrix ADC CPX pod。
- `NS_DNS_FORCE_TCP`：此环境变量强制对 DNS 请求使用 TCP，即使查询是通过 UDP 接收的亦如此。
- `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 **Citrix ADC CPX** 部署为节点上的 **DNS** 缓存

将 Citrix ADC CPX 部署为 Kubernetes 群集的本地 DNS 缓存包括以下任务：

在主节点上：

- 创建使用端点作为 Citrix ADC CPX pod 的 Kubernetes 服务

- 创建用于为 Citrix ADC CPX pod 定义环境变量的 ConfigMap
- 使用 Kubernetes DaemonSet 在 Kubernetes 群集中的每个节点上安排 Citrix ADC CPX pod。

在工作节点上：

- 使用 Citrix ADC CPX 服务的群集 IP 地址修改 kubelet 配置文件，以将 DNS 请求转发到 Citrix ADC CPX。

Kubernetes 主节点上的配置

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

1. 使用 `cpx_dns_svc.yaml` 文件创建将 Citrix ADC CPX pod 作为端点的服务。

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

`cpx_dns_svc.yaml` 文件按如下所示提供：

```
        apiVersion: v1
        kind: Service
        metadata:
          name: cpx-dns-svc
          labels:
            app: cpxd
        spec:
          ports:
          - protocol: UDP
            port: 53
            name: dns
          - protocol: TCP
            port: 53
            name: dns-tcp
          selector:
            app: cpx-daemon
```

2. 获取 Citrix ADC CPX 服务的 IP 地址。

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

3. 获取 Kube DNS 服务的 IP 地址。

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

4. 创建用于为 Citrix ADC CPX pod 定义环境变量的 ConfigMap。这些环境变量用于传递 Citrix ADC 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`。


```
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。

```
  CPX_DNS_SVC_IP: 10.111.95.145
  EULA: "yes"
  KUBE_DNS_SVC_IP: 10.96.0.10
  NS_CPX_LITE: "1"
  NS_DNS_EXT_RESLV_IP: 10.102.217.142
  NS_DNS_MATCH_DOMAIN: citrix.com
  PLATFORM: CP1000
```

下面是一个示例 ConfigMap：

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

5. 在主节点上为 Citrix ADC CPX 创建 Kubernetes DaemonSet。

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

cpx_daemonset.yaml 文件按如下所示提供：

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cpx-daemon
  labels:
        app: cpxd
spec:
```



```
    selector:
         matchLabels:
           app: cpx-daemon
template:
   metadata:
         labels:
           app: cpx-daemon
   spec:
       containers:
       - name: cpxd
        imagePullPolicy: IfNotPresent
        image: localhost:5000/dev/cpx
        volumeMounts:
         - mountPath: /netns/default/
        name: test-vol
        ports:
        - containerPort: 53
   envFrom:
       - configMapRef:
       name: cpx-dns-cache
   securityContext:
   privileged: true
   allowPrivilegeEscalation: true
   capabilities:
    add: ["NET_ADMIN"]
   volumes:
      -  name: test-vol
      hostPath:
      path: /proc/1/ns
      type: Directory
```

Kubernetes 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 pod 能够通过执行以下步骤之一来使用 Citrix ADC 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。

     ```
     --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。

  a) 编辑 kubelet 配置并指定 Citrix ADC CPX 服务的群集 IP 地址以及 --cluster-dns 参数的 kube-dns 服务 IP 地址。

```
root@node:~# cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf | grep KUBELET_DNS_ARGS

Environment="KUBELET_DNS_ARGS=--cluster-dns=10.111.95.145,10.96.0.10 --cluster-domain=cluster.local"
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_DNS_ARGS
```

  b) 使用以下命令重新加载节点的 kubelet:

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

在 **Google Compute Engine** 上部署 **Citrix ADC CPX** 代理

December 13, 2021

此部署指南介绍如何在企业网络中运行了 Citrix ADM 的情况下，在 Google Cloud 的 Google Compute Engine (GCE) 上通过 Docker 部署 Citrix ADC CPX。在此部署中，GCE 上安装的 Citrix ADC CPX 对两台后端服务器进行负载平衡，Citrix ADM 提供许可和分析解决方案。

Citrix ADC 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。Citrix ADM 提供管理、许可和分析解决方案。作为许可服务器，Citrix ADM 向本地或云端运行的 Citrix ADC 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 Citrix ADC 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 Citrix ADC CPX 映像。
4. 创建 CPX 实例。
5. 通过 Citrix ADM 许可 Citrix ADC CPX。
6. 在 Citrix ADC CPX 上配置负载平衡服务，并验证配置。
    a) 安装 NGINX Web 服务器。
    b) 为 Citrix ADC CPX 配置负载平衡，并验证负载是否分布到两个 Web 服务。

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：

```
$ sudo curl  - ssl https://get.docker.com/ | sh
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve
    host: xn--ssl-1n0a
100 17409 100 17409 0 0 21510 0 --:--:-- --:--:-- --:--:-- 21492
apparmor is enabled in the kernel and apparmor utils were already
   installed
+ sudo -E sh -c apt-key add -
+ echo -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQINBFWln24BEADrBl5p99uKh8+rpvqJ48u4eTtjeXAWbslJotmC/CakbNSqOb9o
ddfzRvGVeJVERt/Q/mlvEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
mYdoe/BJ/J/6rHOjq7Omk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyu/jPpWFK28TR8yfVlzYFwibj5WK
dHM7ZTqlA1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccyOi+ULU0z8mHUJ3iEMIrpT
X+80KaN/ZjibfsBOCjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
HOoyC6qVeOgKrRiKxzymj0FIMLru/iFF5pSWcBQB7PYlt8J0G80lAcPr6VCiN+4c
NKv03SdvA69dCOj79PuO9IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHHM1fDeCqkZ
```



```
hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Zaltn0svoGs+wX3Z/H6/8P5anog43U
65c0A+64Jj00rNDr8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+cOMHUU4HA7iM
zHrouAdYeTZeZEQOA7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFNcTKdv+rwARAQAB
tDdEb2NrZXIgUmVsZWFzZSBUb29sIChyZWxlYXNlZG9ja2VyKSA8ZG9ja2VyQGRv
Y2tlci5jb20+iQIcBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4SyoOCV
cnybl18vzwBEcp4VCRbXvHvOXty1gccVIV8/aJqNKgBV97lY3vrpOyiIeB8ETQeg
srxFE7t/Gz0rsLObqfLEHdmn5iBJRkhLfCpzjeOnyB3Z0IJB6UogO/msQVYe5CXJ
l6uwr0AmoiCBLrVlDAktxVh9RWch0l0KZRX2FpHu8h+uM0/zySqIidlYfLa3y5oH
scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdcAHR+bXhA/RW8QHeeMFliWw
7Z2jYHyuHmDnWG2yUrnCqAJTrWV+OfKRIzzJFBs4e88ru5h2ZIXdRepw/+COYj34
LyzxR2cxr2u/xvxwXCkSMe7F4KZAphD+1ws61FhnUMi/PERMYfTFuvPrCkq4gyBj
t3fFpZ2NR/fKW87QOeVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsxrkFQ2jNp
D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaHO
W1zblWy/CFr6VFl963jrjJgag0G6tNtBZLrclZgWhOQpeZZ5Lbvz2ZA5CqRrfAVc
wPNW1fObFIRtqV6vuVluFOPCMAAnOnqR02w9t17iVQjO3oVN0mbQi9vjuExXh1Yo
ScVetiO6LSmlQfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBlAAoJEFyzYeVS
+w0QeH0QAI6btAfYwYPuAjfRUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+l/R08nu0d
sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
NydGFzmeOKbAlrxIMECnSTG/TkFVYO9Ntlv9vSN2BupmTagTRErxLZKnVsWRzp+X

-----END PGP PUBLIC KEY BLOCK-----

OK
+ sudo -E sh -c mkdir -p /etc/apt/sources.list.d
+ dpkg --print-architecture
+ sudo -E sh -c echo deb \\[arch=amd64\\] https://apt.dockerproject.org
    /repo ubuntu-yakkety main > /etc/apt/sources.list.d/docker.list
+ sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-
    engine
Hit:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety InRelease
Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates
    InRelease [102 kB]
Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports
    InRelease [102 kB]
Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted
    Sources [5,376 B]
Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse
    Sources [181 kB]
Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
    Sources [8,044 kB]
Get:7 http://archive.canonical.com/ubuntu yakkety InRelease [11.5 kB]
Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease [102
     kB]
Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease [47.3
     kB]
```



```
55 Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main
      Sources [903 kB]
56 Get:11 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      restricted Sources [2,688 B]
57 Get:12 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      universe Sources [57.9 kB]
58 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
       consider
144 adding your user to the "docker" group with something like:
145
146 sudo usermod -aG docker albert_lee
147
148 Remember that you will have to log out and back in for this to take
       effect.
149
150 WARNING: Adding a user to the "docker" group will grant the ability to
       run
151 containers which can be used to obtain root privileges on the
```



```
152 docker host.
153 Refer to https://docs.docker.com/engine/security/security/#docker-
       daemon-attack-surface
154 for more information.
155
156 $
157
158 \*\*$ sudo docker info\*\*
159 Containers: 0
160 Running: 0
161 Paused: 0
162 Stopped: 0
163 Images: 0
164 Server Version: 17.05.0-ce
165 Storage Driver: aufs
166 Root Dir: /var/lib/docker/aufs
167 Backing Filesystem: extfs
168 Dirs: 0
169 Dirperm1 Supported: true
170 Logging Driver: json-file
171 Cgroup Driver: cgroupfs
172 Plugins:
173 Volume: local
174 Network: bridge host macvlan null overlay
175 Swarm: inactive
176 Runtimes: runc
177 Default Runtime: runc
178 Init Binary: docker-init
179 containerd version: 9048e5e50717ea4497b757314bad98ea3763c145
180 runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228
181 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
```



```
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
$

\*\*$ sudo docker images\*\*
REPOSITORY TAG IMAGE ID CREATED SIZE
$

\*\*$ sudo docker ps\*\*
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
$
<!--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。Citrix ADM 需要此端口才能与 Docker 实例进行通信。

```

 \*\*cd /etc/systemd/system\*\*
 \*\*sudo vi docker-tcp.socket\*\*
 \*\*cat docker-tcp.socket\*\*
[Unit]
\*\*Description=Docker Socket for the API
[Socket]
ListenStream=4243
BindIPv6Only=both
Service=docker.service
[Install]
WantedBy=sockets.target\*\*

$ \*\*sudo systemctl enable docker-tcp.socket\*\*
Created symlink /etc/systemd/system/sockets.target.wants/docker-tcp.
   socket → /etc/systemd/system/docker-tcp.socket.
 \*\*sudo systemctl enable docker.socket\*\*
 \*\*sudo systemctl stop docker\*\*
 \*\*sudo systemctl start docker-tcp.socket\*\*
 \*\*sudo systemctl start docker\*\*
$ \*\*sudo systemctl status docker\*\*
```



```
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 {
```



```
 "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" }

(external)$

<!--NeedCopy-->
```

步骤 **3**：安装 **Citrix ADC CPX** 映像

从 Docker App Store 获取 Citrix ADC 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。

```
$ \*\*sudo docker pull store/citrix/citrixadccpx:13.0-36.29\*\*
13.0-36.29: Pulling from store/citrix/citrixadccpx
4e1f679e8ab4: Pull complete
a3ed95caeb02: Pull complete
2931a926d44b: Pull complete
362cd40c5745: Pull complete
d10118725a7a: Pull complete
1e570419a7e5: Pull complete
d19e06114233: Pull complete
d3230f008ffd: Pull complete
22bdb10a70ec: Pull complete
1a5183d7324d: Pull complete
241868d4ebff: Pull complete
3f963e7ae2fc: Pull complete
fd254cf1ea7c: Pull complete
33689c749176: Pull complete
59c27bad28f5: Pull complete
588f5003e10f: Pull complete
Digest: sha256:31
   a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe
Status: Downloaded newer image for store/citrix/citrixadccpx:13.0-36.29
$

$ \*\*sudo docker images\*\*
REPOSITORY TAG IMAGE ID CREATED SIZE
store/citrix/citrixadccpx:13.0-36.29 6fa57c38803f 3 weeks ago 415MB
$
<!--NeedCopy-->
```


步骤 **4**：创建 **Citrix ADC CPX** 实例

在 Docker 主机上安装 Citrix ADC 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 Citrix ADM 指定 IP 地址：

```
bash-2.05b# \*\*CHOST=${
 1:-localhost }
 \*\*
bash-2.05b# \*\*echo | openssl s_client -connect $CHOST:443 | openssl
   x509 -fingerprint -noout | cut -d'=' -f2\*\*
depth=0 C = US, ST = California, L = San Jose, O = Citrix NetScaler, OU
    = Internal, CN = Test Only Cert
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = California, L = San Jose, O = Citrix NetScaler, OU
    = Internal, CN = Test Only Cert
verify return:1
DONE
24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4
bash-2.05b#

$ \*\*sudo docker run -dt -p 50000:88 -p 5080:80 -p 5022:22 -p 5443:443
    -p 5163:161/udp -e NS_HTTP_PORT=5080 -e NS_HTTPS_PORT=5443 -e
   NS_SSH_PORT=5022 -e NS_SNMP_PORT=5163 -e EULA=yes -e LS_IP=xx.xx.xx.
   xx -e PLATFORM=CP1000 --privileged=true --ulimit core=-1 -e
   NS_MGMT_SERVER=xx.xx.xx.xx:xxxx -e NS_MGMT_FINGER_PRINT=24:AA:8B
   :91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4 --env
   NS_ROUTABLE=false --env HOST=104.199.209.157 store/citrix/
   citrixadccpx:13.0-36.29\*\*
44ca1c6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
$

$ \*\*sudo docker ps\*\*
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
44ca1c6c0907 store/citrix/citrixadccpx:13.0-36.29 "/bin/sh -c 'bash ...
   " 19 seconds ago Up 17 seconds 0.0.0.0:5022->22/tcp,
   0.0.0.0:5080->80/tcp, 0.0.0.0:50000->88/tcp, 0.0.0.0:5163->161/udp,
   0.0.0.0:5443->443/tcp gifted_perlman
$

$ \*\*ssh -p 5022 root@localhost\*\*
root@localhost's password:
Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86_64)

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



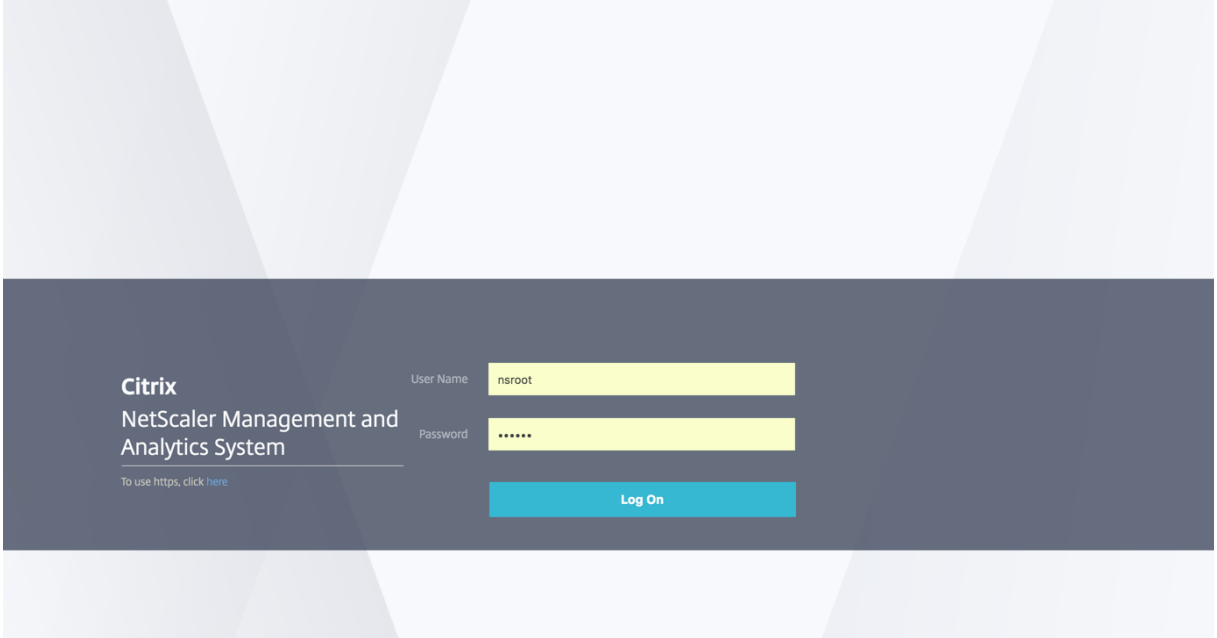
```
Last login: Mon Jun 5 18:58:51 2017 from xx.xx.xx.xx
root@44ca1c6c0907:~#
root@44ca1c6c0907:~#
root@44ca1c6c0907:~# \*\*cli_script.sh 'show ns ip'\*\*
exec: show ns ip
Ipaddress Traffic Domain Type Mode Arp Icmp Vserver State
--------- -------------- ---- ---- --- ---- ------- ------
1) 172.17.0.2 0 NetScaler IP Active Enabled Enabled NA Enabled
2) 192.0.0.1 0 SNIP Active Enabled Enabled NA Enabled
Done
root@44ca1c6c0907:~# \*\*cli_script.sh 'show licenseserver'\*\*
exec: show licenseserver
1) ServerName: xx.xx.xx.xxPort: 27000 Status: 1 Grace: 0 Gptimeleft: 0
Done
root@44ca1c6c0907:~# cli_script.sh 'show capacity'
exec: show capacity
Actualbandwidth: 1000 Platform: CP1000 Unit: Mbps Maxbandwidth: 3000
    Minbandwidth: 20 Instancecount: 0
Done
root@44ca1c6c0907:~#

$ \*\*sudo iptables -t nat -L -n\*\*
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
DOCKER all -- 0.0.0.0/0 !127.0.0.0/8 ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22

Chain DOCKER (2 references)
target prot opt source destination
RETURN all -- 0.0.0.0/0 0.0.0.0/0
```

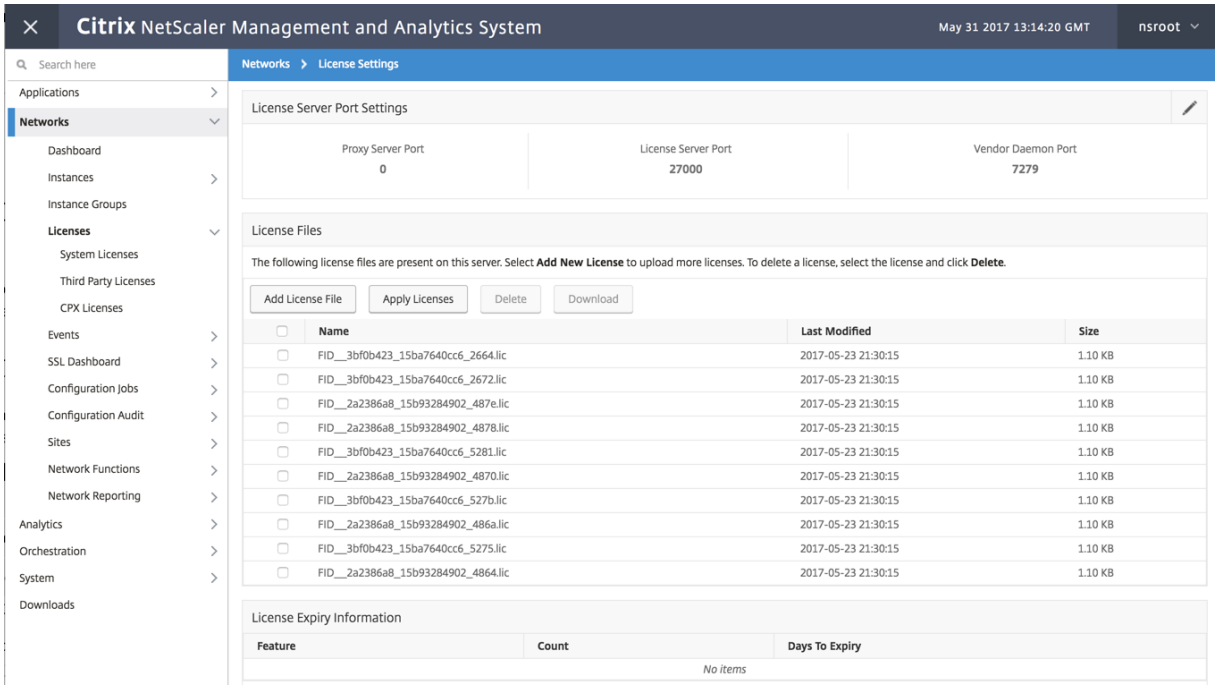


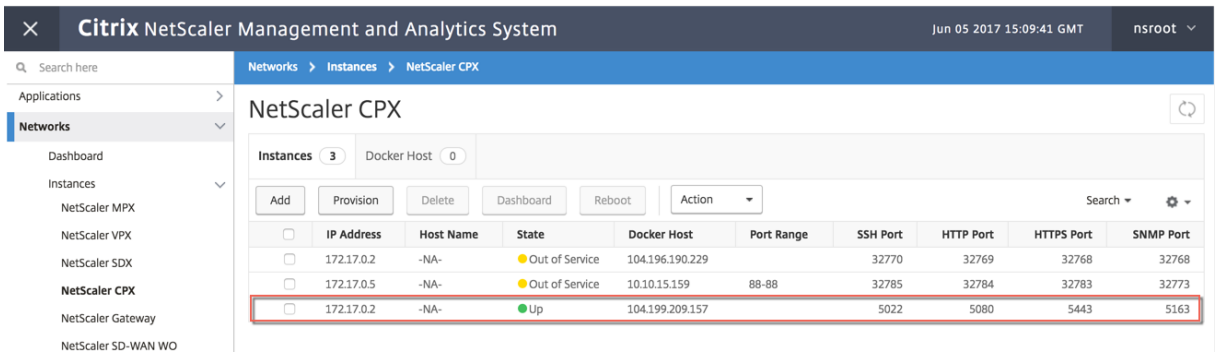
```
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
$
<!--NeedCopy-->
```

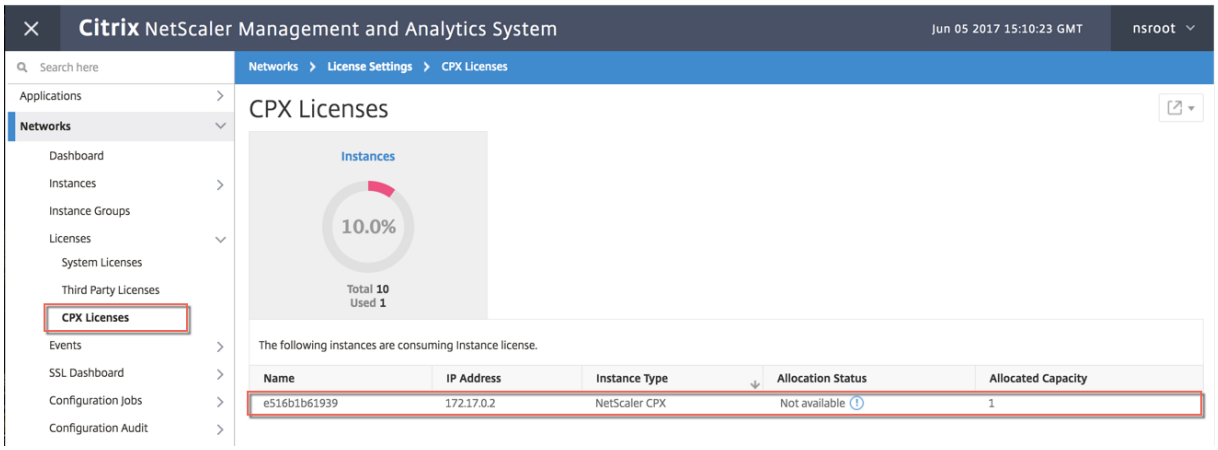
步骤 **5**：通过 **Citrix ADM** 许可 **Citrix ADC CPX**

假设 Citrix ADM 在本地运行，您应该能够验证 Citrix ADC CPX 是否与 MAS 通信并发送信息。下面的图片显示了 Citrix ADC CPX 如何从 Citrix ADM 检索许可证。









步骤 **6**：在 **Citrix ADC CPX** 上配置负载平衡服务，并验证配置

首先，在 Docker 主机上安装 NGINX Web 服务器。然后，在 Citrix ADC CPX 上配置负载平衡以对两台 Web 服务器进行负载平衡，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。

```
$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:41
    ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
Status: Image is up to date for nginx:latest


\*\*$ sudo docker run -d -p 81:80 nginx\*\*
098a77974818f451c052ecd172080a7d45e446239479d9213cd4ea6a3678616f


\*\*$ sudo docker run -d -p 82:80 nginx\*\*
bbdac2920bb4085f70b588292697813e5975389dd546c0512daf45079798db65


\*\*$ sudo iptables -t nat -L -n\*\*
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
DOCKER all -- 0.0.0.0/0 !127.0.0.0/8 ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80
MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80

Chain DOCKER (2 references)
target prot opt source destination
```



```
RETURN all -- 0.0.0.0/0 0.0.0.0/0
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80
$
<!--NeedCopy-->
```

为 **Citrix ADC CPX** 配置负载平衡，并验证负载是否分布到两个 **Web** 服务

```
$ \*\*ssh -p 5022 root@localhost\*\*
root@localhost's password:
Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86_64)

* Documentation: https://www.citrix.com/
Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1
root@44ca1c6c0907:~#
root@44ca1c6c0907:~#
root@44ca1c6c0907:~#
root@44ca1c6c0907:~#
root@44ca1c6c0907:~# \*\*cli_script.sh "add service web1 172.17.0.3
   HTTP 80"\*\*
exec: add service web1 172.17.0.3 HTTP 80
Done
root@44ca1c6c0907:~# \*\*cli_script.sh "add service web2 172.17.0.4
   HTTP 80"\*\*
exec: add service web2 172.17.0.4 HTTP 80
Done
root@44ca1c6c0907:~# \*\*cli_script.sh "add lb vserver cpx-vip HTTP
   172.17.0.2 88"\*\*
exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
Done
root@44ca1c6c0907:~# \*\*cli_script.sh "bind lb vserver cpx-vip web1
   "\*\*
exec: bind lb vserver cpx-vip web1
Done
root@44ca1c6c0907:~# \*\*cli_script.sh "bind lb vserver cpx-vip web2
   "\*\*
exec: bind lb vserver cpx-vip web2
Done
root@44ca1c6c0907:~#
```



```
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 RHIstate: 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 \\<\\!DOCTYPE html\\>
70 \<html\>
```



```
\<head\>
\<title\>Welcome to nginx\!\</title\>
\<style\>
body {

width: 35em;
margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif;
 }

\\</style\\>
\\</head\\>
\<body\>
\<h1\>Welcome to nginx\!\</h1\>
\<p\>If you see this page, the nginx web server is successfully
   installed and
working. Further configuration is required.\\</p\\>

\<p\>For online documentation and support please refer to
\\<a href="http://nginx.org/"\\>nginx.org\\</a\\>.\\<br/\\>
Commercial support is available at
\\<a href="http://nginx.com/"\\>nginx.com\\</a\\>.\\</p\\>

\<p\>\<em\>Thank you for using nginx.\</em\>\</p\>
\\</body\\>
\\</html\\>
(external)$


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

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

                                 Dload  Upload   Total   Spent    Left
    Speed

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

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

```



```
                                 Dload  Upload   Total   Spent    Left
      Speed

100   612  100   612    0     0   1893      0 --:--:-- --:--:--
   --:--:--  1894

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

                                 Dload  Upload   Total   Spent    Left
      Speed

100   612  100   612    0     0   1884      0 --:--:-- --:--:--
   --:--:--  1883

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

                                 Dload  Upload   Total   Spent    Left
      Speed

100   612  100   612    0     0   1917      0 --:--:-- --:--:--
   --:--:--  1924

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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

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



```
                                 Dload  Upload   Total   Spent    Left
      Speed

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

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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

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

                                 Dload  Upload   Total   Spent    Left
      Speed

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


  % Total    % Received % Xferd  Average Speed   Time    Time     Time
```



```
      Current
172
173                                 Dload  Upload   Total   Spent    Left
      Speed
174
175 100   612  100   612    0     0   1862      0 --:--:-- --:--:--
    --:--:--  1860
176
177   % Total    % Received % Xferd  Average Speed   Time    Time     Time
      Current
178
179                                 Dload  Upload   Total   Spent    Left
      Speed
180
181 100   612  100   612    0     0   1806      0 --:--:-- --:--:--
    --:--:--  1810
182
183   % Total    % Received % Xferd  Average Speed   Time    Time     Time
      Current
184
185                                 Dload  Upload   Total   Spent    Left
      Speed
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
    actSvcs
204
205 cpx-vip           172.17.0.2    88         HTTP           UP      100
            2
```



```
206
207
208
209             inactSvcs
210
211 cpx-vip             0
212
213
214
215 Virtual Server Statistics
216
217                                                 Rate (/s)
       Total
218
219 Vserver hits                                            0
       101
220
221 Requests                                                0
       101
222
223 Responses                                               0
                 101
224
225 Request bytes                                           0
       8585
226
227 Response bytes                                          0
       85850
228
229 Total Packets rcvd                                      0
       708
230
231 Total Packets sent                                      0
       408
232
233 Current client connections                             --
        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                              --
```



```
        0
270
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-->
```

Citrix ADC CPX 故障排除

February 21, 2022

本文档介绍了如何对使用 Citrix ADC CPX 时可能会遇到的问题进行故障排除。使用本文档，您可以收集日志以确定原因并应用与 Citrix ADC CPX 的安装和配置有关的一些常见问题的解决方法。

- 我怎样才能查看 Citrix ADC CPX 日志?

  如果使用 `tty:true` 选项部署 Citrix ADC CPX，则可以使用 `kubectl logs` 命令查看 Citrix ADC 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
  ```

  下面是使用 `tty:true` 选项的 Citrix ADC CPX pod 部署的示例：

  ```
  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-->
  ```


可以在 Citrix ADC CPX 文件系统的 */cpx/log/boot.log* 文件中找到更多引导日志。

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

- 我如何从 Citrix ADC CPX 收集技术支持包?

  可以在 Kubernetes 主节点的 shell 界面上运行以下命令来收集 Citrix ADC CPX 技术支持包：

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

  可以在 Citrix ADC 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 Citrix ADC CPX 复制到所需的目标位置。

  示例：

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

- Citrix ADC CPX pod 为什么会在启动时卡住?

  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：

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

  示例：

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

  如果 pod 事件显示容器已启动，则必须检查 pod 日志。

- 如何在 Citrix ADC CPX pod 与 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 Citrix ADC CPX 容器退出核心转储，挂载点上将提供日志和其他重要数据。

可以使用以下任意命令之一在 Citrix ADC CPX pod 与 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：

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

示例：

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

scp：可以使用命令在 Citrix ADC CPX pod 与 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：

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

示例：

```
root@localhost:~# scp nsroot@192.168.29.198:/cpx/log/boot.log /
    tmp/cpx-boot.log
nsroot@192.168.29.198's password:
boot.log
100% 7880     5.1MB/s   00:00
root@localhost:~#
```

- 我如何在 Citrix ADC CPX 上捕获数据包?

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

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

  示例：

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

  同时，运行以下命令开始执行数据包捕获：

  ```
  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：


```
cli_script.sh “stop nstrace”
```

可以在 Citrix ADC CPX 文件系统上的 */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>` 环境变量部署了 Citrix ADC CPX 时，也没有配置许可证服务器?

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

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

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

  示例：

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

- 为什么即使在 Citrix ADC CPX 上成功配置许可证服务器之后，Citrix ADC CPX 上也没有配置池许可证?

  验证 */cpx/log/boot.log* 文件中的许可证配置日志。还可以在 Citrix ADC CPX pod 的 shell 界面上使用以下命令验证 Citrix ADC CPX 上已配置的池许可证：

  ```
  cli_script.sh “show capacity”
  ```

  示例：

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

  此外，请务必在许可证服务器中上载所需的许可证文件。使用以下命令在 Citrix ADC CPX 上成功配置之后，还可以验证许可证服务器上的可用许可证。请在 Citrix ADC CPX pod 的 shell 界面上运行以下命令：

  ```
  cli_script.sh “sh licenseserverpool”
  ```

  示例：


```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
    licenseserverpool"
exec: show licenseserverpool
    Instance Total                    : 5
    Instance Available                : 4
    Standard Bandwidth Total          : 0 Mbps
    Standard Bandwidth Availabe       : 0 Mbps
    Enterprise Bandwidth Total        : 0 Mbps
    Enterprise Bandwidth Available    : 0 Mbps
    Platinum Bandwidth Total          : 10.00 Gbps
    Platinum Bandwidth Available      : 9.99 Gbps
    CP1000 Instance Total             : 100
    CP1000 Instance Available         : 100
  Done
<!--NeedCopy-->
```

- 为什么 NITRO API 调用会获得来自 Citrix ADC CPX 的 *Connection Refused*（连接被拒绝）响应?

  自 Citrix ADC 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 Citrix ADC CPX 的 NITRO 端口暴露在 pod 上。可以在 Citrix ADC CPX 容器部分中运行 `kubectl describe` 命令以查看 Citrix ADC CPX 容器的暴露和映射的端口：

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

  示例：

  ```
      ng472 | grep -i port
         Ports:          80/TCP, 443/TCP, 9080/TCP, 9443/TCP
         Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP
           NS_HTTP_PORT:            9080
           NS_HTTPS_PORT:           9443
         Port:          <none>
         Host Port:     <none>
           NS_PORT:                 80
  <!--NeedCopy-->
  ```

- 为什么即使在没有流量或流量很少时，Citrix ADC CPX 中的 NSPPE 进程会占用大部分 CPU 使用率?

  如果 Citrix ADC CPX 是通过 `CPX_CONFIG=' { "YIELD":" NO" } '` 环境变量部署的，即使在没有流量或流量很少时，NSPPE 进程也会占用 100%的 CPU 使用率。如果您希望 NSPPE 进程不占用 CPU 使用率，则必须在不使用 `CPX_CONFIG=' { "YIELD":" NO" }` 环境变量的情况下部署 Citrix ADC CPX。默认情况下，CPX 中的 NSPPE 进程配置为不占用 CPU 使用率。

- 为什么即使是通过在 Citrix ADM 中注册所需的环境变量部署的，Citrix ADC CPX 也没有在 Citrix ADM 中列出?

可以在 Citrix ADC CPX 文件系统上的 */cpx/log/boot.log* 文件中找到 Citrix ADC CPX 注册到 Citrix ADM 的相关日志。

可以使用 `ping` 命令从 Citrix ADC CPX pod 验证 Citrix ADM IP 地址的可访问性。此外，请务必为 Citrix ADC CPX 容器配置 Citrix 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`：Citrix ADC CPX pod IP 地址不能从外部路由。
- `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 Citrix ADC CPX 上的 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:
         nsroot123"
  7
  8  exec: show ns ip
  9
  10 Ipaddress        Traffic Domain         Type             Mode
           Arp     Icmp     Vserver  State
  11 --------------   ----                   ----             ---
         ----     -------  ------   -----
  12 172.17.0.3       0                  NetScaler IP     Active
         Enabled  Enabled  NA       Enabled
  13 192.0.0.1        0                  SNIP             Active
         Enabled  Enabled  NA       Enabled
  ```



```
14    Done
```

- 为什么 `root` 和 `nsroot` 用户通过 SSH 加密 Citrix ADC CPX 的数据失败?

  自版本 13.0-64.35 起，Citrix ADC CPX 会生成默认密码并为 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 Citrix ADC CPX 的文件系统上的 `/var/deviceinfo/random_id` 中找到 SSH 用户的密码。

Locations
Corporate Headquarters | 851 Cypress Creek Road Fort Lauderdale, FL 33309, United States
Silicon Valley | 4988 Great America Parkway Santa Clara, CA 95054, United States