

Citrix ADC CPX 12.1

Contents

About	5
Supported Features	5
Architecture and Traffic Flow	6
How Single IP Address Works on	7
Traffic Flow for Requests Originating from the {{page.netscaler-cpx-short}} Instance	8
Traffic Flow for Requests Originating from the External Network	9
{{page.netscaler-cpx-short}} licensing	11
Types of {{page.netscaler-cpx-short}} licensing	11
Supported pooled capacity for {{page.netscaler-cpx-short}} instances	12
How does {{page.netscaler-cpx-short}} licensing work?	12
{{page.netscaler-cpx-short}} Express	13
{{page.netscaler-cpx-short}} licensing models	13
Provision bandwidth-based and vCPU-based licensing from Citrix ADM service for	14
Provision Bandwidth based licensing and vCPU based licensing for {{page.netscaler-cpx-short}} from Citrix ADM on-premises	16
Commands for cleaning the deployments	18
Deploying a {{page.netscaler-cpx-short}} Instance in Docker	19
Prerequisites	19
Downloading {{page.netscaler-cpx-short}} Image from Quay	20
Deploying the {{page.netscaler-cpx-short}} Instance Using the docker run Command	20
Deploying {{page.netscaler-cpx-short}} Instances by Using Docker Compose	25
Adding {{page.netscaler-cpx-short}} Instances to Citrix ADM	29
Prerequisites	29
Adding {{page.netscaler-cpx-short}} instances to Citrix ADM using environment variables	31
Configuring	32
Configuring a {{page.netscaler-cpx-short}} Instance by Using the Command Line Interface	32

Configuring a {{page.netScaler-cpx-short}} Instance by Using the NITRO API	33
Configuring a {{page.netScaler-cpx-short}} Instance by Using Jobs	33
Configuring Log Streaming on a {{page.netScaler-cpx-short}} Instance	35
Configuring {{page.netScaler-cpx-short}} Using a Configuration File	37
Structure of the static configuration file	37
Configuring Docker Logging Drivers	40
Dynamic Routing support in	41
Configure BGP based Route Health Injection using REST API on	42
Upgrading a NetScaler CPX™ Instance	45
Prerequisites	45
To upgrade a NetScaler® CPX instance	46
Using Wildcard Virtual Servers in NetScaler CPX™ Instance	48
Deploy {{page.netScaler-cpx-short}} as a Proxy to Enable East-West Traffic Flow	50
Deploy {{page.netScaler-cpx-short}} in a Single Host Network	55
Deploy {{page.netScaler-cpx-short}} in a Multi-Host Network	57
Topology 1: {{page.netScaler-cpx-short}} and Backend Servers on Same Host; Client on a Different Network	57
Topology 2: {{page.netScaler-cpx-short}} with Physical Servers and Client	60
Topology 3: {{page.netScaler-cpx-short}} and Servers Provisioned on Different Hosts	62
Deploy {{page.netScaler-cpx-short}} with direct access to the network	65
Configure {{page.netScaler-cpx-short}} in Kubernetes Using ConfigMaps	67
Deploy {{page.netScaler-cpx-short}} Proxy on Google Compute Engine	73
Prerequisites	73
Configuration Steps	73
Step 1: Install Docker on a GCE VM	74
Step 2: Configure Remote API Communication with the Docker Instance	85
Step 3: Install {{page.netScaler-cpx-short}} Image	88
Step 4: Create a CPX Instance	90

Step 5: License {{page.netscaler-cpx-short}} Through NetScaler MAS	94
Step 6: Configure Load Balancing Services on {{page.netscaler-cpx-short}} and Verify the Configuration	96
Install NGINX Web Servers	96
Configure CPX for load balancing and verify distribution of load to both web services	98
{{page.netscaler-cpx-short}} troubleshooting	113

About

{{page.netscaler-cpx-short}} is a container-based application delivery controller that can be provisioned on a Docker host. {{page.netscaler-cpx-short}} enables customers to leverage Docker engine capabilities and use NetScaler load balancing and traffic management features for container-based applications. You can deploy one or more {{page.netscaler-cpx-short}} instances as standalone instances on a Docker host.

A {{page.netscaler-cpx-short}} instance provides throughput of up to 1 Gbps.

This document assumes that you are familiar with Docker and how it works. For information about Docker, see the Docker documentation at <https://docs.docker.com>.

Supported Features

{{page.netscaler-cpx-short}} supports the following features:

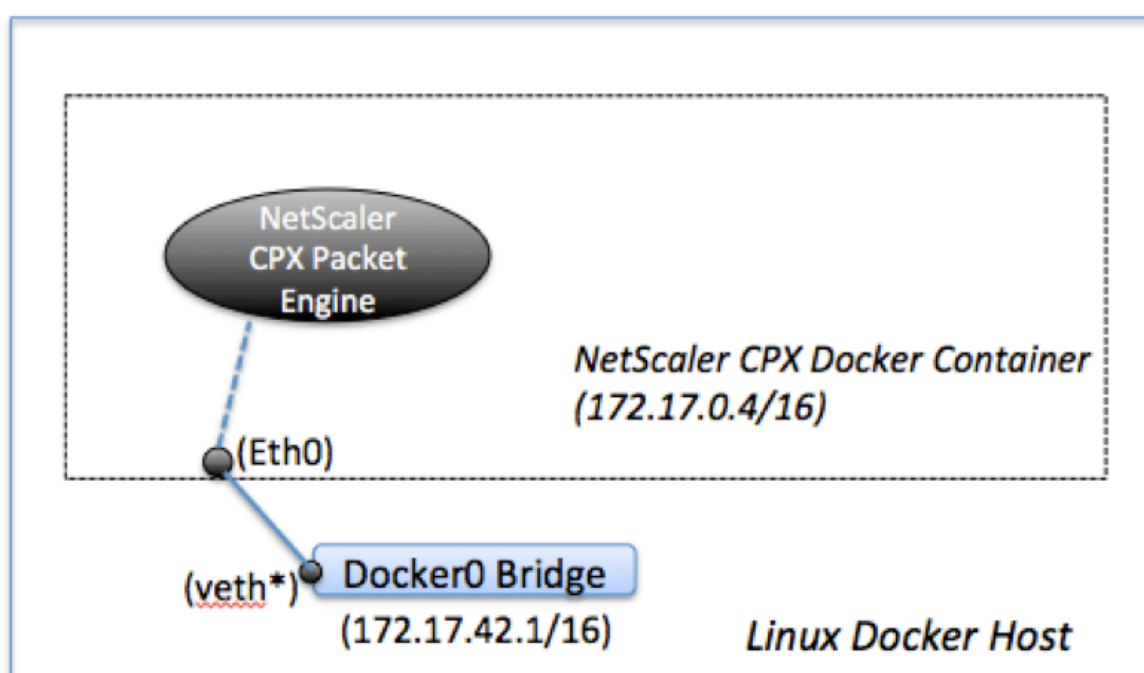
- Application availability
 - L4 load balancing and L7 content switching
 - SSL Offloading
 - IPv6 protocol translation
- Application security
 - L7 rewrite and responder
- Simple manageability
 - Web logging
 - AppFlow®

Architecture and Traffic Flow

When you provision a `{{page.netscaler-cpx-short}}` instance on a Docker host, the Docker engine creates a virtual interface, `eth0`, on the CPX instance. This `eth0` interface is directly connected to a virtual interface (`veth*`) on the `docker0` bridge. The Docker engine also assigns an IP address to the `{{page.netscaler-cpx-short}}` instance in the network `172.17.0.0/16`.

The default gateway for the CPX instance is the IP address of the `docker0` bridge, which means that any communication with the `{{page.netscaler-cpx-short}}` instance is done through the Docker network. All incoming traffic received from the `docker0` bridge is received by the `eth0` interface on the `{{page.netscaler-cpx-short}}` instance and processed by the `{{page.netscaler-cpx-short}}` packet engine.

The following figure illustrates the architecture of a `{{page.netscaler-cpx-short}}` instance on a Docker host.



How Single IP Address Works on

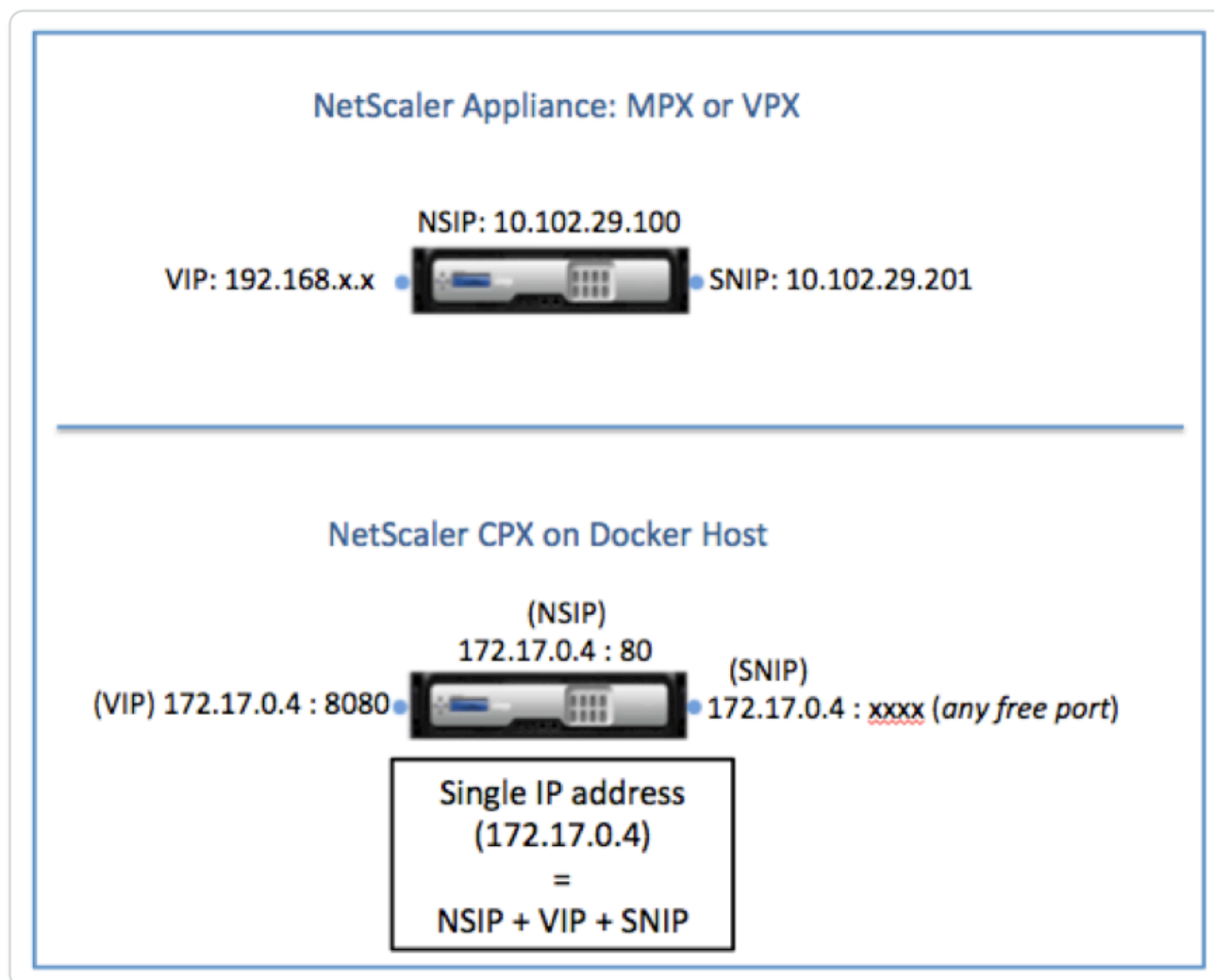
A regular NetScaler MPX™ or VPX appliance requires at least three IP addresses to function:

- Management IP address called the NetScaler® IP (NSIP) address
- Subnet IP (SNIP) address for communicating with the server farm
- Virtual server IP (VIP) address(es) for accepting client requests

A `{{page.netscaler-cpx-short}}` instance operates with one single IP address that is used for management as well as for data traffic.

During provisioning, only one private IP address (single IP address) is assigned to a `{{page.netscaler-cpx-short}}` instance by the Docker engine. The three IP functions of a NetScaler instance are multiplexed onto one IP address. This single IP address uses different port numbers to function as the NSIP, SNIP, and VIP(s).

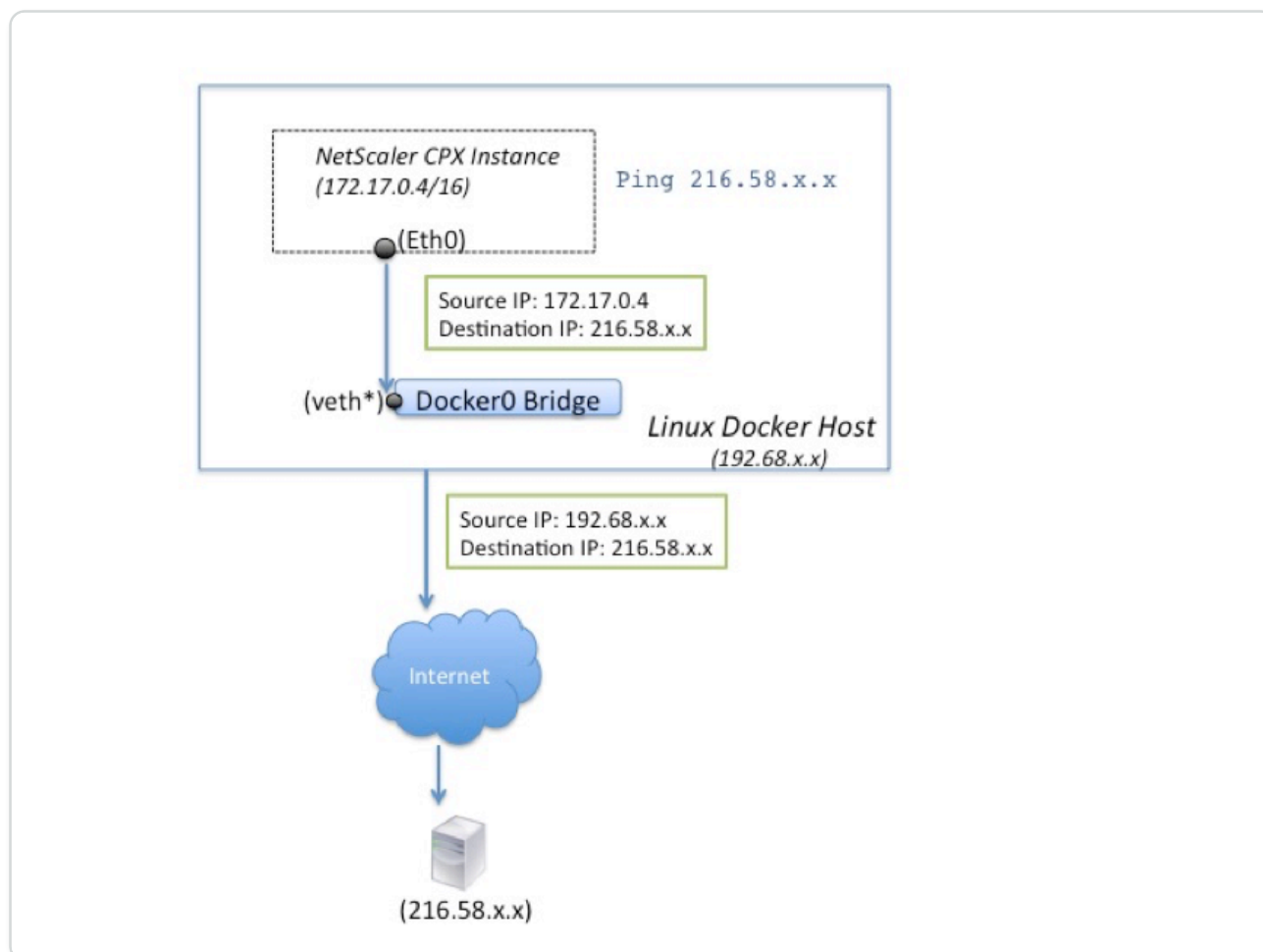
The following image illustrates how a single IP address is used to perform the functions of NSIP, SNIP, and VIP(s).



Traffic Flow for Requests Originating from the `{{page.netscaler-cpx-short}}` Instance

Docker implicitly configures IP tables and a NAT rule to direct traffic originating from the `{{page.netscaler-cpx-short}}` instance to the `docker0` IP address.

The following figure illustrates how a ping request originating from a `{{page.netscaler-cpx-short}}` instance reaches the destination.



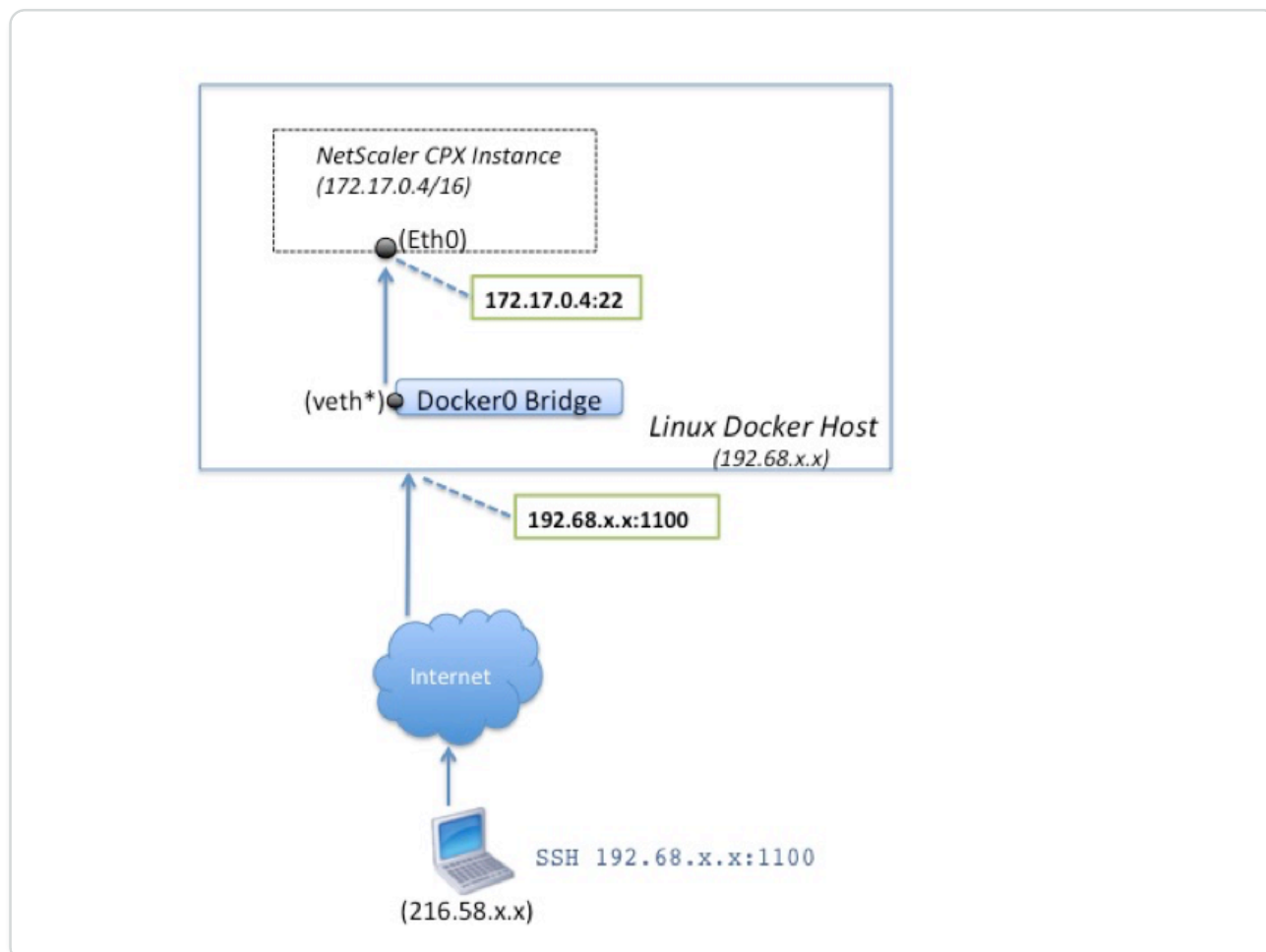
In this example, the ping request is sent by the packet engine on eth0 interface with source IP address as the {{page.netscaler-cpx-short}} IP address (172.17.0.4). The Docker host, then, performs network address translation (NAT) to add the host IP address (192.68.x.x) as the source IP address and sends the request to the destination (216.58.x.x). The response from the destination IP address follows the same path in reverse. The Docker host performs NAT on the response and forwards the response to the {{page.netscaler-cpx-short}} instance on the eth0 interface.

Traffic Flow for Requests Originating from the External Network

To enable external communication, while provisioning {{page.netscaler-cpx-short}}, you have to set parameters such that Docker exposes certain ports such as 80, 22, and any other port you want. If you have not set any port to be exposed during provisioning, then you have to configure NAT rules on the Docker host to make these ports available.

The client request that originates from the Internet is received by the Docker host, which then performs port address translation (PAT) to map the public IP address and port to the single IP address and port of the {{page.netscaler-cpx-short}} instance, and forwards the traffic to the instance.

The following figure shows how the Docker host performs port address translation to direct traffic to the {{page.netscaler-cpx-short}} single IP address and port.



In this example, the Docker host IP address is 192.68.x.x and the single IP address of the {{page.netscaler-cpx-short}} instance is 172.17.0.4. The SSH port 22 of {{page.netscaler-cpx-short}} instance is mapped to port 1100 on the Docker host. The SSH request from the client is received on IP address 192.68.x.x at port 1100. The Docker host performs port address translation to map this address and port to the single IP address 172.17.0.4 on port 22 and forwards the client request.

{{page.netscaler-cpx-short}} licensing

{{page.netscaler-cpx-short}} is a container-based application delivery controller that can be provisioned on a Docker host to load balance microservice based applications. You need licensed CPX for better performance of application delivery. {{page.netscaler-cpx-short}} supports pool licensing. Citrix ADM can act as your license server to license your {{page.netscaler-cpx-short}} instances.

Citrix® ADM is available both on-premises and a cloud service as well. You can use the Citrix ADM to manage pooled capacity licenses for all Citrix ADC form factors.

For information about Citrix ADM on-premises, see [Citrix ADM on-premises](#). For information about Citrix ADM service, see [Citrix ADM service](#).

Types of {{page.netscaler-cpx-short}} licensing

{{page.netscaler-cpx-short}} supports bandwidth and virtual CPU (core) pool licensing for on-prem and cloud based deployments.

Bandwidth pool: {{page.netscaler-cpx-short}} licenses can be allocated based on the bandwidth consumption by the instances. You can use pooled licensing to maximize the bandwidth utilization by ensuring the necessary bandwidth allocation to an instance and not more than its requirement. For more information about the bandwidth pool, see [Bandwidth based licensing](#).

vCPU pool: In the virtual CPU-usage-based licensing, the license specifies the number of CPUs that a particular {{page.netscaler-cpx-short}} instance is entitled to. So, the {{page.netscaler-cpx-short}} can check out licenses for only the number of virtual CPUs from the license server. {{page.netscaler-cpx-short}} checks out licenses depending on the number of CPUs running in the system. For more information about the vCPU pool, see [Citrix ADC virtual CPU licensing](#).

Supported pooled capacity for {{page.netscaler-cpx-short}} instances

PRODUCT	MAXIMUM BANDWIDTH H	MINIMUM BANDWIDTH H	MINIMUM INSTANCES	MAXIMUM INSTANCES	MINIMUM BANDWIDTH H UNIT
{{page.netscaler-cpx-short}}	40000 Note: It depends on CPU frequency, generation, and so on.	20 Mbps	1	16	10 Mbps

Note: Citrix is currently working on a {{page.netscaler-cpx-short}} consumption based or pay-as-you-grow based licensing model for public cloud-based offerings. Once ready, it will be available on the public cloud market place to consume.

How does {{page.netscaler-cpx-short}} licensing work?

{{page.netscaler-cpx-short}} pooled capacity: A common license pool from which your {{page.netscaler-cpx-short}} instance can check out one instance license and only as much bandwidth as it needs. When the instance no longer requires these resources, it checks them back in to the common pool, making the resources available to other instances which need these licenses.

{{page.netscaler-cpx-short}} check-in and check-out licensing: Citrix ADM allocates licenses {{page.netscaler-cpx-short}} instances on demand. A {{page.netscaler-cpx-short}} instance can check out the license from the Citrix ADM when a {{page.netscaler-cpx-short}} instance is provisioned and check back in its license to Citrix ADM when an instance is destroyed.

{{page.netscaler-cpx-short}} behavior: A single {{page.netscaler-cpx-short}} instance checking-out up to 1 Gbps throughput, checks-out only from the instance pool and not from the bandwidth license pool. {{page.netscaler-cpx-short}} operates in this way up to the 1 Gbps of bandwidth

utilization. For example, if a CPX instance consumes a 200 Mbps bandwidth, it uses the instance pool of license, instead of the bandwidth pool. However, if a {{page.netscaler-cpx-short}} instance consumes 1200 Mbps of throughput, the first 1000 Mbps is utilized from the instance pool and the remaining 200 Mbps is consumed from the bandwidth pool.

{{page.netscaler-cpx-short}} Express

{{page.netscaler-cpx-short}} Express is a software edition that is free-of-cost for on-premises and cloud deployments. When you download {{page.netscaler-cpx-short}} instance from the [Quay](#) repository, this is the default capacity available for POCs which do not require a license file and it comes with the following features:

- 20 Mbps bandwidth
- Maximum 250 SSL sessions
- 20 Mbps SSL throughput

You must license your {{page.netscaler-cpx-short}} instance to upgrade for better performance and production deployments.

{{page.netscaler-cpx-short}} licensing models

Citrix offers a range of product licensing models for {{page.netscaler-cpx-short}} to meet your organization's requirements. You can select options such as vCPU or bandwidth and on-premises or cloud.

Based on your requirements, you can choose any of the following models:

- Bandwidth based licensing for {{page.netscaler-cpx-short}} from ADM service
- vCPU based licensing for {{page.netscaler-cpx-short}} from ADM service
- Bandwidth based licensing for {{page.netscaler-cpx-short}} from ADM on-premises
- vCPU based licensing for {{page.netscaler-cpx-short}} from ADM on-premises

Provision bandwidth-based and vCPU-based licensing from Citrix ADM service for

Perform the following steps to provision bandwidth-based license and vCPU-based license for {{page.netscaler-cpx-short}} from Citrix ADM service.

1. Set up Citrix ADM.

Ensure that the Citrix ADM service setup is operational with the {{page.adm-ads-agent-short}}. You must have a Citrix ADM service and {{page.adm-ads-agent-short}} account for {{page.netscaler-cpx-short}} licensing to be functional. For information about setting up Citrix ADM service and {{page.adm-ads-agent-short}}, see [Citrix ADM service](#).

Note: In this procedure, a hypervisor (on-premises) {{page.adm-ads-agent-short}} setup is used. In the following image, `10.106.102.199` is the on-premises agent used for licensing {{page.netscaler-cpx-short}}.



2. Add Citrix ADC instance license pool to Citrix ADM service.

It is assumed that you have a pool of bandwidth licenses available for ADM service. For information about uploading a license file to Citrix ADM, see [Configure pooled capacity](#). In the following image, `CNS_INST_200CC_Retail.lic` is used as the bandwidth and instance license pool.



3. Deploy {{page.netscaler-cpx-short}} instance in Kubernetes cluster. Ensure that the following environment variables are added to the {{page.netscaler-cpx-short}} YAML file to license the {{page.netscaler-cpx-short}} instance.

For the bandwidth-based licensing from the Citrix ADM service, specify the following environment variables in the YAML file:

- name: "LS_IP" value: "10.105.158.166" //ADM agent IP as mentioned in step 1
- name: "LS_PORT" value: "27000" // port on which ADM license server listens
- name: "BANDWIDTH" value: "3000" //capacity in Mbps wants to allocate to CPX

For the vCPU-based licensing from the Citrix ADM service, specify the following environment variables in the YAML file:

- name: "LS_IP" value: "10.102.216.173" //ADM agent IP as mentioned in step 1
- name: "LS_PORT" value: "27000" // port on which ADM license server listens to
- name: "CPX_CORES" value: "4" // number of core you want to allocate

4. Download the `cpk-bandwidth-license-adm-service.yaml` file using the following command:

```
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. Deploy the edited YAML in the Kubernetes cluster using the following command:

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

6. Log in to {{page.netscaler-cpx-short}} for verifying instancing information by using the following command:

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

7. To view the licensing information for the given {{page.netscaler-cpx-short}} instance, run the following commands:

```
cli_script.sh "show licenseserver"

cli_script.sh "show capacity"
```

You can track the allocated bandwidth and vCPU capacity in the ADM service portal.

Provision Bandwidth based licensing and vCPU based licensing for {{page.netscaler-cpx-short}} from Citrix ADM on-premises

Perform the following steps to provision bandwidth-based and vCPU-based to {{page.netscaler-cpx-short}} from Citrix ADM on-premises.

1. Set up Citrix ADM.

Ensure that the ADM on-premises setup is ready. Make sure that Citrix ADM on-premises with or without ADM agent deployment for {{page.netscaler-cpx-short}} licensing is functioning.

For information about setting up Citrix ADM on-premises and {{page.adm-ads-agent-short}}, see [Citrix ADM service](#).

Note: In this example, an in-built ADM agent with ADM on-premises is used. In the following image, you can see that there is no agent deployed.



2. Add Citrix ADC instance license pool to ADM on-premises.

It is assumed that you have a pool of bandwidth license available for ADM on-premises. To know about uploading a license file to Citrix ADM, see [Licensing](#). In the following image, `CNS_INST_200CC_Retail.lic` is used as the bandwidth and instance license pool.



In the following image, CP1000 is used as the vCPU license pool.



3. Deploy {{page.netscaler-cpx-short}} instance in Kubernetes cluster. Ensure that the following environment variables are added to the {{page.netscaler-cpx-short}} YAML file to license the {{page.netscaler-cpx-short}} instance.

For the bandwidth-based licensing from Citrix ADM on-premises, specify the following environment variables in the YAML file:

- name: "LS_IP" value: "10.105.158.144" // ADM on-prem instance IP, If you have deployed ADM agent, then this is your agent IP address as described in the step 1

- name: "LS_PORT" value: "27000" // port on which ADM license server listens
- name: "BANDWIDTH" value: "3000" //capacity in Mbps wants to allocate to CPX

For the vCPU-based licensing from Citrix ADM on-premises, specify the following environment variables in the YAML file:

- name: "LS_IP" value: "10.105.158.144" // ADM on-prem instance IP, If you have ADM agent deployment then this will be your agent IP as described in step 1
- name: "LS_PORT" value: "27000" // port on which ADM license server listens to
- name: "CPX_CORES" value: "4" // the number of cores that you want to allocate

4. Download the `cpx-bandwidth-license-adm-onprem.yaml` file using the following command:

```
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. Deploy the edited YAML in the Kubernetes cluster using the following command:

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

6. Log in to {{page.netscaler-cpx-short}} for verifying instancing information by using the following command:

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

7. To view the licensing information for the {{page.netscaler-cpx-short}} instance, run the following commands:

```
cli_script.sh "show licenseserver"

cli_script.sh "show capacity"
```

You can track the allocated bandwidth and vCPU capacity in the ADM on-premises portal.

Commands for cleaning the deployments

You can use the following commands to clean the various YAML deployments:

```
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
```

Deploying a {{page.netscaler-cpx-short}} Instance in Docker

{{page.netscaler-cpx-short}} instances are available as a Docker image file in the Quay container registry. To deploy an instance, download the {{page.netscaler-cpx-short}} image from the Quay container registry and then deploy the instance by using the `docker run` command or the Docker compose tool.

Prerequisites

Make sure that:

- Docker host system has at least:
 - 1 CPU
 - 2 GB RAM

NOTE:

For better {{page.netscaler-cpx-short}} performance, you can define the number of processing engines that you want the {{page.netscaler-cpx-short}} instance to start. For every additional processing engine, you add, make sure that the Docker host contains the equivalent number of vCPUs and amount of memory in GB. For example, if you want to add 4 processing engines, the Docker host must contain 4 vCPUs and 4 GB of memory.

- Docker host system is running Linux Ubuntu version 14.04 or later.
- Docker version 1.12 is installed on the host system. For information about Docker installation on Linux, see the [Docker Documentation](#).
- Docker host has Internet connectivity.

Note: {{page.netscaler-cpx-short}} has issues while running on ubuntu version 16.04.5, kernel version 4.4.0-131-generic. So, it is not recommended to run {{page.netscaler-cpx-short}} on ubuntu version 16.04.5 kernel version 4.4.0-131-generic.

Downloading {{page.netscaler-cpx-short}} Image from Quay

You can download {{page.netscaler-cpx-short}} image from the Quay container registry using the `docker pull` command and deploy it on your environment. Use the following command to download the {{page.netscaler-cpx-short}} image from the Quay container registry:

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

In this command, tag specifies the Citrix the {{page.netscaler-cpx-short}} image.

For example, if you want to download the version 12.1-51.16, then use the following command:

```
docker pull quay.io/citrix/citrix-k8s-cpx-ingress:12.1-51.16
```

Use the following command to verify if {{page.netscaler-cpx-short}} image is installed in docker images:

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

quay.io/citrix/citrix-k8s-cpx-ingress	12.1-51.16
952a04e73101	2 months ago 469 MB

Note: It is recommended to use the latest {{page.netscaler-cpx-short}} image from the Quay container registry for availing the latest features of {{page.netscaler-cpx-short}}.

Deploying the {{page.netscaler-cpx-short}} Instance Using the docker run Command

On the host, you can install a {{page.netscaler-cpx-short}} instance in the Docker container by using the {{page.netscaler-cpx-short}} Docker image that you loaded onto the host. Using the

`docker run` command, install the {{page.netscaler-cpx-short}} instance with the default {{page.netscaler-cpx-short}} configuration.

IMPORTANT:

If you have downloaded {{page.netscaler-cpx-short}} Express from <https://www.citrix.com/products/netscaler-adc/cpx-express.html>, make sure that you read and understand the End User License Agreement (EULA) available at: <https://www.citrix.com/products/netscaler-adc/cpx-express.html> and accept the EULA while deploying the {{page.netscaler-cpx-short}} instance.

Install the {{page.netscaler-cpx-short}} instance on the Docker container by using the following **`docker run`** command:

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

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"No"}' -e
LS_IP=10.102.38.134 -e PLATFORM=CP1000 -v /var/cpx:/cpx --name cpx_host
cpx:12.1-48.xx
```

This example creates a container named mycpx based on the {{page.netscaler-cpx-short}} Docker image.

The `-P` parameter is mandatory. It tells Docker to map the ports exposed in the container by the {{page.netscaler-cpx-short}} Docker image. That means map ports 9080, 22, 9443, and 161/UDP, to the ports on the Docker host that are randomly selected from the user defined range. This mapping is done to avoid conflicts. If you later create multiple {{page.netscaler-cpx-short}} containers on the same Docker host. The port mappings are dynamic and are set each time the container is started or restarted. The ports are used as follows:

- 9080 is used for HTTP
- 9443 is used for HTTPS
- 22 used for SSH

- 161/UDP is used for SNMP.

If you want static port mappings, use the `-p` parameter to set them manually.

The `--privileged=true` option is used to run the container in privileged mode. If you are running the `{{page.netscaler-cpx-short}}` with multiple cores then you need to provide all the system privileges to the `{{page.netscaler-cpx-short}}`. If you want to run the `{{page.netscaler-cpx-short}}` with a single core then instead of this option you must use the `--cap-add=NET_ADMIN` option. The `--cap-add=NET_ADMIN` option enables you to run the `{{page.netscaler-cpx-short}}` container with full network privileges.

The `**--net=host` is a standard docker run command option that specifies that the container is running in the host network stack and has access to all the network devices.

NOTE

If you are running `{{page.netscaler-cpx-short}}` in bridge or none network, ignore this option.

The `-e NS_NETMODE="HOST"` is a `{{page.netscaler-cpx-short}}` specific environment variable that allows you to specify that the `{{page.netscaler-cpx-short}}` is started in host mode. Once `{{page.netscaler-cpx-short}}` starts in host mode it configures four default iptable rules on the host machine for management access to the `{{page.netscaler-cpx-short}}`. It uses the following ports:

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

If you want to specify different ports, you can use the following environment variables:

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

NOTE

If you are running {{page.netscaler-cpx-short}} in bridge or none network, ignore this environment variable.

The `-e CPX_CORES` is an optional {{page.netscaler-cpx-short}} specific environment variable. You can use it to improve the performance of the {{page.netscaler-cpx-short}} instance by defining the number of processing engines that you want the {{page.netscaler-cpx-short}} container to start.

NOTE

For every additional processing engine you add, make sure that the Docker host contains the equivalent number of vCPUs and amount of memory in GB. For example, if you want to add 4 processing engines, then the Docker host must contain 4 vCPUs and 4 GB of memory.

The `-e EULA = yes` is a mandatory {{page.netscaler-cpx-short}} specific environment variable, which is required to verify that you have read and understand the End User License Agreement (EULA) available at: <https://www.citrix.com/products/netscaler-adc/cpx-express.html>.

The `-e PLATFORM=CP1000` parameter specifies the {{page.netscaler-cpx-short}} license type.

If you are running Docker in a host network, you can assign dedicated network interfaces to the {{page.netscaler-cpx-short}} container using the `-e CPX_NW_DEV` environment variable. You need to define the network interfaces separated by a whitespace. The network interfaces that you define are held by the {{page.netscaler-cpx-short}} container until you uninstall the {{page.netscaler-cpx-short}} container. When the {{page.netscaler-cpx-short}} container is provisioned all the assigned network interfaces are added to the NetScaler networking namespace.

NOTE

If you are running {{page.netscaler-cpx-short}} in a bridge network you may change the container network, such as, configure another network connection to the container or remove an existing network. Then make sure that you restart the {{page.netscaler-cpx-short}} container to use the updated network.

```
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:12.0-53.x
```

The `-e CPX_CONFIG` is a `{{page.netscaler-cpx-short}}` specific environment variable that enables you to control the throughput performance of the `{{page.netscaler-cpx-short}}` container. When the `{{page.netscaler-cpx-short}}` does not receive any incoming traffic to process, it yields the CPU during this idle time, hence resulting in low throughput performance. You can use the

`CPX_CONFIG` environment variable to control the throughput performance of the `{{page.netscaler-cpx-short}}` container in such scenarios. You need to provide the following values to the

`CPX_CONFIG` environment variable in JSON format:

- If you want the `{{page.netscaler-cpx-short}}` container to yield CPU in idle scenarios, define `{"YIELD" : "Yes"}`
- If you want the `{{page.netscaler-cpx-short}}` container to avoid yielding the CPU in idle scenarios so that you can get high throughput performance, define `{"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:12.0-51.x
```

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD": "Yes"}' -e PLATFORM=CP1000
--name cpx_host cpx:12.0-51.xx
```

The `-v` parameter is an optional parameter that specifies the mount point of the `{{page.netscaler-cpx-short}}` mount directory, `/cpx`. A mount point is a directory on the host, in which you mount the `/cpx` directory. The `/cpx` directory stores the logs, configuration files, SSL certificates, and core dump files. In the example, the mount point is `/var/cpx` and the `{{page.netscaler-cpx-short}}` mount directory is `/cpx`.

If you purchased a license or have an evaluation license, you can upload the license to a license server and specify the license server location with the docker run command, by using the `-e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT>` parameter. In this case, you do not need to accept the 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:12.0-51.xx
```

Where:

- **LS_IP_ADDRESS** is the IP address of the license server.
- **LS_PORT** is the port of the license server.

You can view the images running on your system and the ports mapped to the standard ports by using the command: `docker ps`

Deploying {{page.netscaler-cpx-short}} Instances by Using Docker Compose

You can use the Compose tool of Docker to provision a single {{page.netscaler-cpx-short}} instance or multiple {{page.netscaler-cpx-short}} instances. To provision {{page.netscaler-cpx-short}} instances by using Compose, you must first write a compose file. This file specifies the {{page.netscaler-cpx-short}} image, the ports that you want to open for the {{page.netscaler-cpx-short}} instance, and the privileges for your {{page.netscaler-cpx-short}} instance.

IMPORTANT

Make sure that you have installed Docker Compose tool on the host.

To provision multiple {{page.netscaler-cpx-short}} instances:

1. Write a compose file, where:

- **<service-name>** is the name of the service you want to provision.
- **image:<repository>:<tag>** denotes the repository and the versions of the {{page.netscaler-cpx-short}} image.
- **privileged: true** provides all root privileges to the {{page.netscaler-cpx-short}} instance.
- **cap_add** provides network privileges to the {{page.netscaler-cpx-short}} instance.

- **<host_directory_path>** denotes the directory on the docker host that you want to mount for the `netScaler-cpx-short` instance.
- **<number_processing_engine>** is the number of processing engines that you want the `netScaler-cpx-short` instance to start. For every additional processing engine, make sure that the Docker host contains the equivalent number of vCPUs and amount of memory in GB. For example, if you want to add 4 processing engines, then the Docker host must contain 4 vCPUs and 4 GB of memory.

The compose file generally follows a format similar to:

```
<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"}'

```
CPX_0:
```

```
container_name: CPX_0
```

```
image: cpx:12.0-53.xx
```

```
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

If you want to provision a single {{page.netscaler-cpx-short}} instance, you must add the following line to the compose file: `container_name:<name_of_container>`

Run the following command to provision multiple {{page.netscaler-cpx-short}} instances: `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`

If you want to provision a single {{page.netscaler-cpx-short}} instance, run the following command:
`docker-compose -f <compose_file_name> up -d`

Adding {{page.netscaler-cpx-short}} Instances to Citrix ADM

You must add the {{page.netscaler-cpx-short}} instances installed on a Docker host to Citrix Application Delivery Management (ADM) software if you want to manage and monitor these instances.

You can add instances either while setting up ADM for the first time or at a later time.

To add instances, you must create an instance profile and specify either the host name or IP address of each instance, or a range of IP addresses. This instance profile contains the user name and password of the instance(s) that you want to add to Citrix® ADM. For each instance type, a default profile is available. For example, the ns-root-profile is the default profile for Citrix ADC instances. This profile is defined by the default ADC administrator credentials. If you have changed the default admin credentials of your instances, you can define custom instance profiles for those instances. If you change the credentials of an instance after the instance is discovered, you must edit the instance profile or create a new profile, and then rediscover the instance.

Prerequisites

Make sure that you have:

- Installed the Citrix ADM software on Citrix XenServer. For more information, see [Citrix ADM Documentation](#).
- Installed the {{page.netscaler-cpx-short}} instances on a Docker host.

To add {{page.netscaler-cpx-short}} instances to ADM:

1. In a web browser, type the IP address of the **Citrix Application Delivery Management** (for example, `http://192.168.100.1`).
2. In the **User Name** and **Password** fields, enter the administrator credentials. The default administrator credentials are **nsroot** and **nsroot**.
3. Navigate to **Networks** \> **Instances** > **Citrix ADC** and click **CPX** tab.
4. Click **Add** to add new CPX instances in Citrix ADM.

5. The **Add {{page.netscaler-cpx-short}}** page opens. Enter the values for the following parameters:
- You can add CPX instances by providing either the reachable IP address of the CPX instance or the IP address of the Docker container where the CPX instance is hosted.
 - Select the profile of the CPX instance.
 - Select the site where the instances is to be deployed.
 - Select the agent.
 - As an option, you can enter the key-value pair to the instance. Adding key-value pair makes it easy for you to search for the instance later.

Add Citrix ADC CPX

☒ Enter Device IP Address ☐ Import from file

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

Routable IP/ Docker IP*

172.31.32.161 ?

Profile Name*

Docker-profile ▼ Add Edit

Site*

Ohio-site ▼ Add Edit

Agent

Click to select >

Tags

Key Value +

OK Close

6. Click **OK**.

NOTE

If you want to rediscover an instance, navigate to **Networks \> Instances \> Citrix ADC > CPX**, select the instance you want to rediscover, and then from the **Select Action** drop-down list, click **Rediscover**.

Adding {{page.netscaler-cpx-short}} instances to Citrix ADM using environment variables

You can also add the {{page.netscaler-cpx-short}} instances to Citrix ADM using environment variables. To add instances, you must configure the following environment variables for the {{page.netscaler-cpx-short}} instance.

- `NS_MGMT_SERVER` - ADM IP address/FQDN
- `HOST` - Node IP address
- `NS_HTTP_PORT` - Mapped HTTP port on node
- `NS_HTTPS_PORT` - Mapped HTTPS port on node
- `NS_SSH_PORT` - Mapped SSH port on node
- `NS_SNMP_PORT` - Mapped SNMP port on node
- `NS_ROUTABLE` - ({{page.netscaler-cpx-short}} pod IP address is not routable from outside.)

The following is an example `docker run` command for adding a {{page.netscaler-cpx-short}} instance to Citrix ADM.

```
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
```

Configuring

You can configure a `{{page.netscaler-cpx-short}}` instance by accessing the CLI prompt through the Linux Docker host, or by using NetScaler NITRO APIs.

Configuring a `{{page.netscaler-cpx-short}}` Instance by Using the Command Line Interface

Access the Docker host and log on to the SSH prompt of the instance as shown in the following figure. The default administrator credentials to log on to a `{{page.netscaler-cpx-short}}` instance are `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:~#
```

Type the following command to use the command line prompt of the instance to run CLI commands: `cli_script.sh "<command>"`

Example:

```
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

To log out of the instance prompt, type `logout`.

Configuring a {{page.netscaler-cpx-short}} Instance by Using the NITRO API

You can use the NetScaler NITRO API to configure {{page.netscaler-cpx-short}} instances.

To configure {{page.netscaler-cpx-short}} instances by using the Nitro API, in a web browser, type:

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

To retrieve statistics by using the Nitro API, in a web browser, type:

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

For more information about using the NITRO API, see [REST Web Services](#). For {{page.netscaler-cpx-short}}, use `CPX IP address:port` where `netscaler-ip-address` is mentioned.

Configuring a {{page.netscaler-cpx-short}} Instance by Using Jobs

You can configure {{page.netscaler-cpx-short}} instances by creating and executing jobs in Citrix ADM. You can use the configurations from configuration templates, extract configurations available on other devices, and use configurations saved in text files. You can also record configurations done by using the configuration utility of another instances. Citrix ADM then displays the corresponding CLI commands for you to use on your {{page.netscaler-cpx-short}} instance. After you select the configuration, you must select {{page.netscaler-cpx-short}} instances on which you want to load the configuration, specify the variable values, and execute the job.

To configure {{page.netscaler-cpx-short}} instances by using Jobs:

1. Log on to Citrix® ADM by using the administrative credentials.
2. Navigate to **Networks** \> **Configuration Jobs**, and then click **Create Job**.
3. Specify the required values, and select the configuration source. You can also type the commands you want to run.

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

Job Name*
cpx-single-host

Instance Type*
NetScaler

Configuration Editor

Configuration Source
Configuration Template

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

LBVariablesTemplate

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

4. Select the {{page.netscaler-cpx-short}} instances on which you want to run the configuration and click **Next**.

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

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

Add Instances | Delete

	IP Address	Name
☐	172.17.0.150	10.102.31.190

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

5. Specify the execution settings and click **Finish** to run the commands on the {{page.netscaler-cpx-short}} instance. If you want to save the configuration and run it later, click **Save and Exit**.

Select Configuration Select Instances Specify Variable Values Execute

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

On Command Failure*
 Ignore error and continue

Execution Mode*
 Now

Execution Settings

☒ Execute in Sequence
☐ Execute in Parallel

Receive Execution Report Through
☐ Email

Cancel Back Finish Save and Exit

Configuring Log Streaming on a {{page.netscaler-cpx-short}} Instance

You can configure log streaming on a {{page.netscaler-cpx-short}} instance to collect webpage performance data, flow and user-session level information, and database information required for application performance monitoring and analytics. These data records are sent to Citrix ADM where you can view real-time and historical reports for all your applications.

To configure log streaming on a {{page.netscaler-cpx-short}} instance, you must first enable the AppFlow feature and the `ulfd` daemon on the {{page.netscaler-cpx-short}} instance. When you enable the `ulfd` daemon, you must specify the IP address of Citrix ADM where you want to monitor the real-time and historical reports. You must then configure an AppFlow collector, action, policy, and bind the AppFlow policy globally.

The `ulfd` daemon sends all the flow records to Citrix ADM in a single unified logging format.

You can configure log streaming either by using the command line interface of a {{page.netscaler-cpx-short}} instance, or by using the Jobs feature in Citrix ADM.

Before you can monitor the records, you must add the {{page.netscaler-cpx-short}} instance to Citrix ADM. For more information about adding a {{page.netscaler-cpx-short}} instance to Citrix ADM, see [Installing a {{page.netscaler-cpx-short}} Instance by Using Citrix ADM](#).

To configure log streaming on a {{page.netscaler-cpx-short}} Instance:

1. Enable the AppFlow feature by running the following command: `enable ns feature AppFlow`
2. Enable the ulfd daemon by using the following command: `set ns param -ulfd ENABLED -loggerip <NUMS_IP_Address>`
3. Run the following commands to configure an AppFlow® collector, action, and policy, and to bind the policy globally:

```
add appflow collector <name> -IPAddress <ipaddress>`

set appflow param -templateRefresh 3600 -httpUrl ENABLED -httpCookie
ENABLED -httpReferer ENABLED -httpMethod ENABLED -httpHost ENABLED
-httpUserAgent ENABLED -httpContentType ENABLED -httpAuthorization ENABLED
-httpVia ENABLED -httpXForwardedFor ENABLED -httpLocation ENABLED
-httpSetCookie ENABLED -httpSetCookie2 ENABLED -connectionChaining ENABLED
-httpDomain Enabled

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

add appflow policy <name> true <action>

bind appflow global <policyName> <priority> [<gotoPriorityExpression
[-type <type>]
```

NOTE:

You must configure the AppFlow collector with a dummy IP address.

Configuring `page.netscaler-cpx-short` Using a Configuration File

Instead of using command line interface (`cli_script.sh`), NITRO API, or NetScaler Management and Analytics (MAS) configuration jobs to configure the `page.netscaler-cpx-short`, you can configure the `page.netscaler-cpx-short` using a static configuration file while deploying the `page.netscaler-cpx-short` instance.

You can provide a static configuration file as an input file while deploying the `page.netscaler-cpx-short` container. During `page.netscaler-cpx-short` container startup, the container is configured based on the configuration specified in the static configuration file. This configuration includes NetScaler-specific configuration and bash shell commands that you can dynamically run on the `page.netscaler-cpx-short` container.

Structure of the static configuration file

As mentioned above, when `page.netscaler-cpx-short` is deployed, it is configured based on the configurations specified in the static configuration file.

The static configuration file is a `.conf` file that includes two tags, `#NetScaler Commands` and `#Shell Commands`. Under the `#NetScaler Commands` tag, you must add all the NetScaler commands to configure the NetScaler-specific configuration on `page.netscaler-cpx-short`. Under the `#Shell Commands` tag, you must add the shell commands that you want to run on `page.netscaler-cpx-short`.

During the `page.netscaler-cpx-short` container deployment, the NetScaler commands and shell commands are run on the container in the order specified in the configuration file.

IMPORTANT:

- The tags can be repeated multiple times in the configuration file.
- The tags are not case-sensitive.
- The configuration file must be present in the `/etc` directory as `cpx.conf` file in the container's file system.
- The configuration file can also include comments. You must add a `"#"` character before your comments.
- If there are failure scenarios while deploying the `{{page.netscaler-cpx-short}}` container with the configuration file, the failures are logged in the `ns.log` file in the container.
- When you reboot the `{{page.netscaler-cpx-short}}` container the configuration file is reapplied on the container.

#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
```

To install a {{page.netscaler-cpx-short}} container and to dynamically configure the {{page.netscaler-cpx-short}} container based on a configuration file, mount the static configuration file using the `-v` option in the `docker run` command:

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -v /tmp/  
cpx.conf:/etc/cpx.conf --name mycpx store/citrix/netscalercpx:12.1-48.13
```

Configuring Docker Logging Drivers

Docker includes logging mechanisms called “logging drivers” to help you get information from the running containers. You can configure a `{{page.netscaler-cpx-short}}` container to forward logs that it generates to the docker logging drivers. For more information on docker logging drivers, see <https://docs.docker.com/config/containers/logging/configure/>.

By default, all logs generated by the `{{page.netscaler-cpx-short}}` container are stored in `/cpx/log/ns.log` file on the docker host. When you start the `{{page.netscaler-cpx-short}}` container using the docker run command, you can configure it to forward all the generated logs to a docker logging driver using the `--log-driver` option. If the logging driver has configurable parameters, you can set them using the `--log-opt <NAME>=<VALUE>` option.

In the following example, the `{{page.netscaler-cpx-short}}` container is configured to forward all the generated logs using syslog as logging driver.

```
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/netscalercpx:12.1-48.13
```

Similarly, in the following example the `{{page.netscaler-cpx-short}}` container is configured to forward all the generated logs using Splunk as logging driver.

```
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/netscalercpx:12.1-48.13
```

Dynamic Routing support in

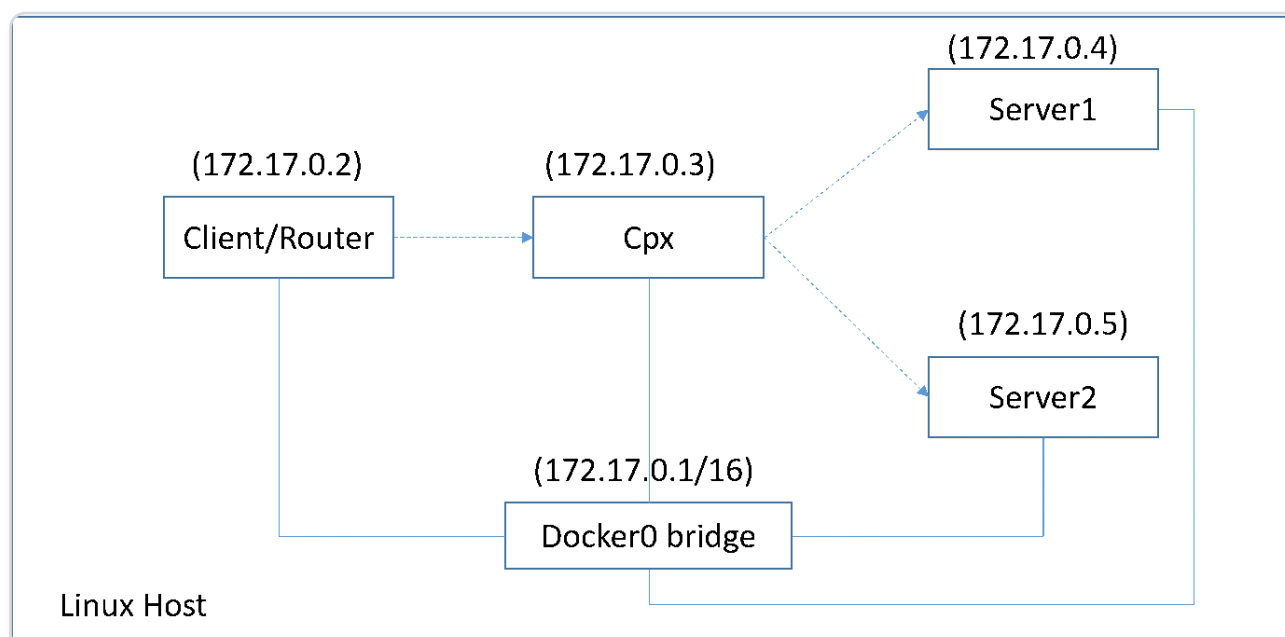
{{page.netscaler-cpx-short}} supports the BGP dynamic routing protocol. The key objective of the dynamic routing protocol is to advertise the virtual server's IP address based on the health of the services, bound to the virtual server. It helps an upstream router to choose the best among multiple routes to a topographically distributed virtual server.

Note: The BGP dynamic routing protocol is supported from build 52.15 of Citrix ADC 12.1 release and later releases.

For information about the non-default password in {{page.netscaler-cpx-short}}, see the [Support for using a non-default password in {{page.netscaler-cpx-short}}](#) section in the [Configuring {{page.netscaler-cpx-short}}](#) document.

In a single host network, the client, the servers, and the {{page.netscaler-cpx-short}} instance are deployed as containers on the same Docker host. All the containers are connected through the docker0 bridge. In this environment, the {{page.netscaler-cpx-short}} instance acts as a proxy for the applications provisioned as containers on the same Docker host. For information about {{page.netscaler-cpx-short}} host networking mode deployment, see [Host networking mode](#).

The following figure illustrates the single host topology.



In this topology, virtual servers are configured and advertised (based on the health of services) to the upstream network or router using BGP.

Perform the following steps to configure BGP on {{page.netscaler-cpx-short}} in single Docker host with the bridge-networking mode.

Configure BGP based Route Health Injection using REST API on

1. Create a container from the {{page.netscaler-cpx-short}} image using the following command:

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

For example:

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

2. Log in to the container using the following command:

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

3. Enable the BGP feature using the following command:

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

4. Obtain the NSIP using the `show ns ip` command:

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

5. Add the virtual server using the following command:

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

6. Add services and bind services to the virtual server.

7. Enable `hostroute` for the VIP using the following command:

```
cli_script.sh "set ns ip &lt;VIP> -hostroute enabled"
```

Log out from the container and send BGP NITRO commands from the host to the NSIP on the port 9080.

8. Configure the BGP router:

For example, if you want to configure:

```
router bgp 100

Neighbour 172.17.0.2 remote-as 101

Redistribute kernel
```

Specify the command as the following:

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

9. Install the learnt BGP routes into the PE using the following NITRO command:

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

10. Verify the BGP adjacency state using the following NITRO command:

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

Sample output:

```
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. Verify that the routes learnt through BGP are installed in the packet engine with the following command:

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

12. Save the configuration using the following command:

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

The dynamic routing configuration is saved in the `/nsconfig/ZebOS.conf` file.

Upgrading a NetScaler CPX™ Instance

You can upgrade a NetScaler CPX instance by shutting it down, installing the latest version on the same mount point, and then deleting the old instance. A mount point is a directory into which you mount the **/cpx** directory on the host.

For example, to mount the **/cpx** directory of the existing NetScaler CPX instance in the host's **/var/cpx** directory, the mount point is **/var/cpx** and the NetScaler CPX mount directory is **/cpx** as shown below:

```
root@ubuntu:~# docker run -dt -e EULA=yes --name mycpx -v /var/cpx:/cpx --ulimit core=-1 cpx:12.1-48.xx
```

Prerequisites

Ensure that you have:

- Details of the host directory in which you mounted the existing NetScaler CPX instance's **/cpx** directory. You can use the `docker inspect <containerName>` command, where `<containerName>` is the name of the NetScaler CPX container, to display information about the host directory.

The output of the command provides the details of the container configurations, including the volumes. In the **"Mounts"** entry, the **"Source"** subentry shows the location of the host directory on the host.

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

- Download the latest NetScaler CPX Docker image file from <https://www.microloadbalancer.com/get-it-now> and then load the NetScaler CPX Docker image. To load the image, navigate to the directory in which you saved the Docker image file. Use the

`docker load -i <image_name>` command to load the image. After the NetScaler CPX image is loaded, you can enter the `docker images` command to display information about the image:

```
root@ubuntu:~# docker load -i cpx-12.0-41.10.gz
```

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

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
cpx	12.0-41.10	2e97aadf918b	43 hours ago	414.5 MB

To upgrade a NetScaler® CPX instance

1. Stop the existing NetScaler CPX instance by entering the `docker stop <containerName>` command, where `<containerName>` is the name of the NetScaler CPX instance.

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

mycpx
```

2. Using the `docker run` command, deploy the latest NetScaler CPX instance from the NetScaler CPX image that you loaded onto the host. Ensure that you deploy the instance at the same mount point (for example, `/var/cpx:/cpx`) that you used for the existing NetScaler CPX instance.

```
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:12.0-41.10
```

You can enter the `docker ps` command so that you can verify that the deployed NetScaler CPX instance is the latest version.

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

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ead12ec4e965	cpx:12.0-41.10	"/bin/sh -c 'bash -C "	5 seconds ago	Up 5 seconds	22/tcp, 80/tcp, 443/tcp, 161/udp	latestcpx

3. After verifying that you deployed the correct NetScaler CPX instance, enter the **`docker rm`** `<containerName>` command to delete the older instance.

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

mycpx

Using Wildcard Virtual Servers in NetScaler CPX™ Instance

When you provision a NetScaler® instance, only one private IP address (single IP address) is assigned to a NetScaler CPX instance by the Docker engine. The three IP functions of a NetScaler instance are multiplexed onto one IP address. This single IP address uses different port numbers to function as the NSIP, SNIP, and VIPs.

The single IP address that is assigned by the Docker engine is dynamic. Add the Load Balancing (LB) or Content Switching (CS) virtual servers using the single IP address or using 127.0.0.1 IP address. The virtual servers created using 127.0.0.1 is called as Wildcard Virtual Servers. By default, when you create a wildcard virtual server, the NetScaler CPX replaces the assigned IP address of the wildcard virtual server. The assigned IP address is 127.0.0.1, with the NSIP assigned to the NetScaler CPX instance by the Docker engine.

In high-availability NetScaler CPX deployments, you can add wildcard virtual servers on one of the NetScaler CPX instances and copy the **ns.conf** file from the instance to other NetScaler CPX instances in the deployment and ensure that the NetScaler configuration is consistent across all the NetScaler CPX instances in the deployment instead of identifying the single IP address assigned by the Docker engine to the NetScaler instance and create the LB or CS virtual servers based on the single IP address on all the NetScaler CPX instances in the deployment.

Points to Note:

- Ensure that the port number that you assign to the wildcard virtual server is not used by any other virtual server in the deployment.
- Wildcard virtual server addition fails if the port number that you assign to the wildcard virtual server is already in use by the internal services.
- The wildcard virtual server does not support the * character.

To create a wildcard load balancing virtual server, at the command prompt, enter the following command:

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

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

To create a wildcard content switching virtual server, at the command prompt, enter the following command:

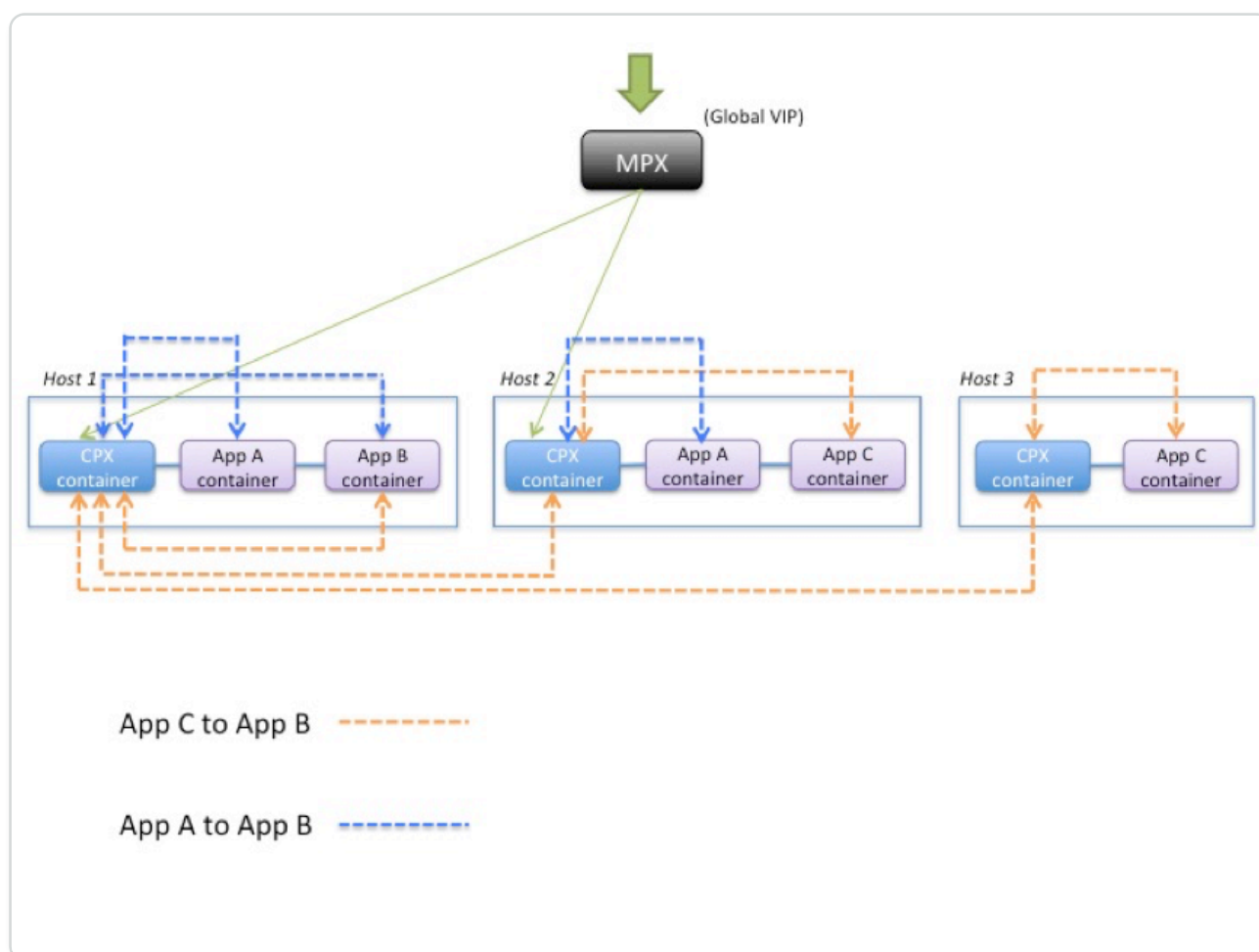
```
add cs vserver <name> <serviceType> 127.0.0.1 <port>
```

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

Deploy {{page.netscaler-cpx-short}} as a Proxy to Enable East-West Traffic Flow

In this deployment, the {{page.netscaler-cpx-short}} instance acts as a proxy to enable communication between application containers residing on multiple hosts. The {{page.netscaler-cpx-short}} instance is provisioned along with the applications in multiple hosts and provides the shortest path for communication.

The following image illustrates traffic flow between two applications through the {{page.netscaler-cpx-short}} instances.

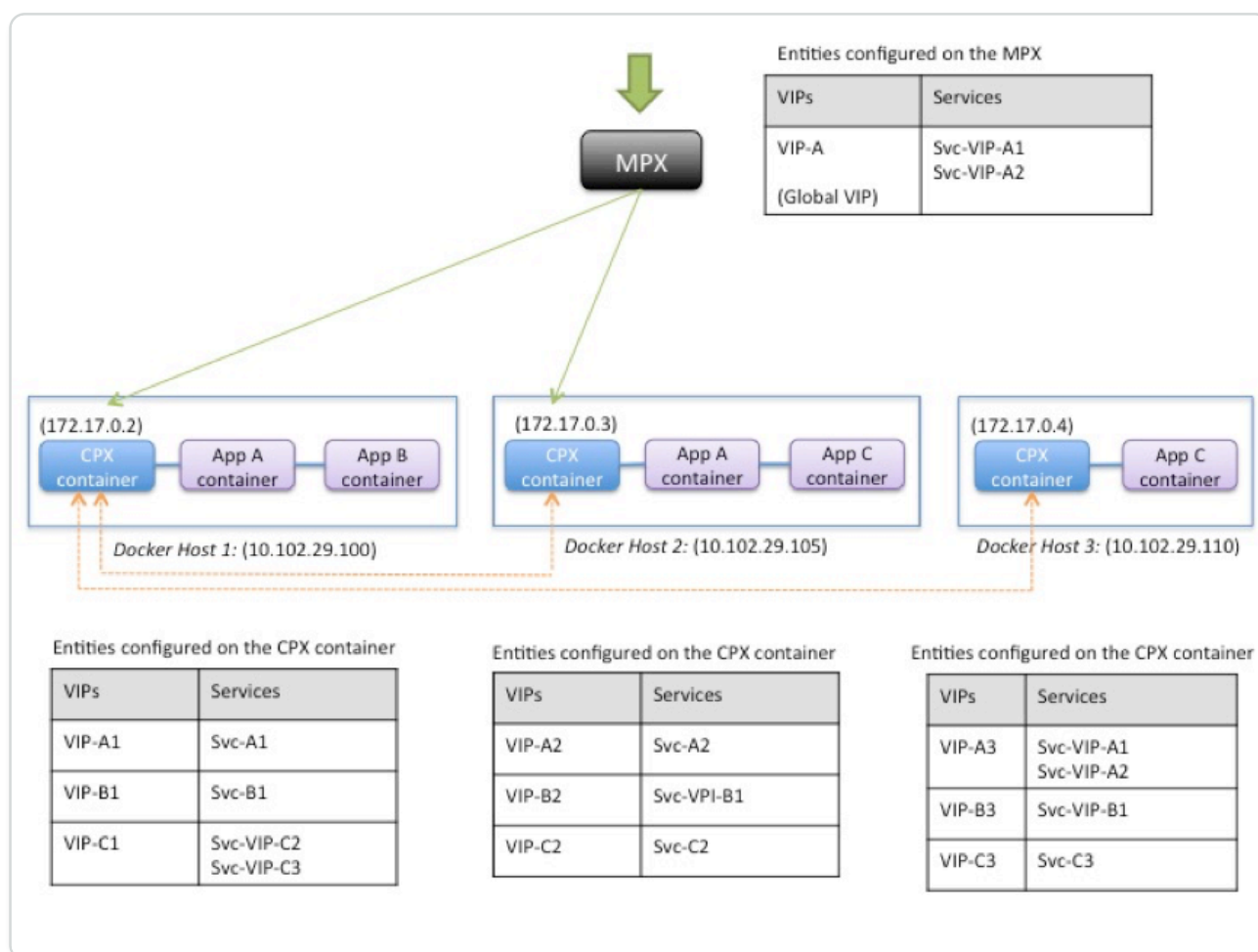


This image shows traffic flow between application C and application B and between application A and application B. When app C (in any of the hosts) sends a request to B, the request is first received on the {{page.netscaler-cpx-short}} container on the same host as app C. Then, the {{page.netscaler-cpx-short}} container passes the traffic to the {{page.netscaler-cpx-short}} container on Host 1, which then routes the traffic to App B.

container hosted on the same host as app B, and then the traffic is forwarded to app B. A similar traffic path is followed when app A sends request to app B.

In this example, a NetScaler MPX is also deployed to allow traffic to the applications from the Internet through a global VIP. The traffic from the NetScaler MPX is received on the `{{page.netscaler-cpx-short}}` containers, which then distributes the traffic across the application containers.

The following diagram illustrates this topology with the configurations that need to be set for communication to happen.



The following table lists the IP addresses and ports that are configured on the `{{page.netscaler-cpx-short}}` instances in this example configuration.

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

To configure this example scenario, run the following command at the Linux shell prompt while creating the `{{page.netscaler-cpx-short}}` container on all three Docker hosts:

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

Run the following commands either by using the Jobs feature in NetScaler® MAS or by using NITRO APIs.

On `{{page.netscaler-cpx-short}}` instance on Docker Host 1:

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

add service svc-A1 10.102.29.100 HTTP 80

bind lb vserver VIP-A1 svc-A1

add lb vserver VIP-B1 HTTP 172.17.0.2 30001

add service svc-B1 10.102.29.100 HTTP 90

bind lb vserver VIP-B1 svc-B1

add lb vserver VIP-C1 HTTP 172.17.0.2 30002
```

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

add service svc-VIP-C3 10.102.29.110 HTTP 30002

bind lb vserver VIP-C1 svc-VIP-C2

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

On the {{page.netscaler-cpx-short}} instance on Docker host 2:

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

add service svc-A2 10.102.29.105 HTTP 80

bind lb vserver VIP-A2 svc-A2

add lb vserver VIP-B2 HTTP 172.17.0.3 30001

add service svc-VIP-B1 10.102.29.100 HTTP 30001

bind lb vserver VIP-B2 svc-VIP-B1

add lb vserver VIP-C2 HTTP 172.17.0.3 30002

add service svc-C2 10.102.29.105 HTTP 70

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

On the {{page.netscaler-cpx-short}} instance on Docker host 3:

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

add service svc-VIP-A1 10.102.29.100 HTTP 30000

add service svc-VIP-A2 10.102.29.105 HTTP 30000

bind lb vserver VIP-A3 svc-VIP-A1

bind lb vserver VIP-A3 svc-VIP-A2

add lb vserver VIP-B3 HTTP 172.17.0.4 30001
```

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

bind lb vserver VIP-B3 svc-VIP-B1

add lb vserver VIP-C3 HTTP 172.17.0.4 30002

add service svc-C3 10.102.29.110 HTTP 70

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

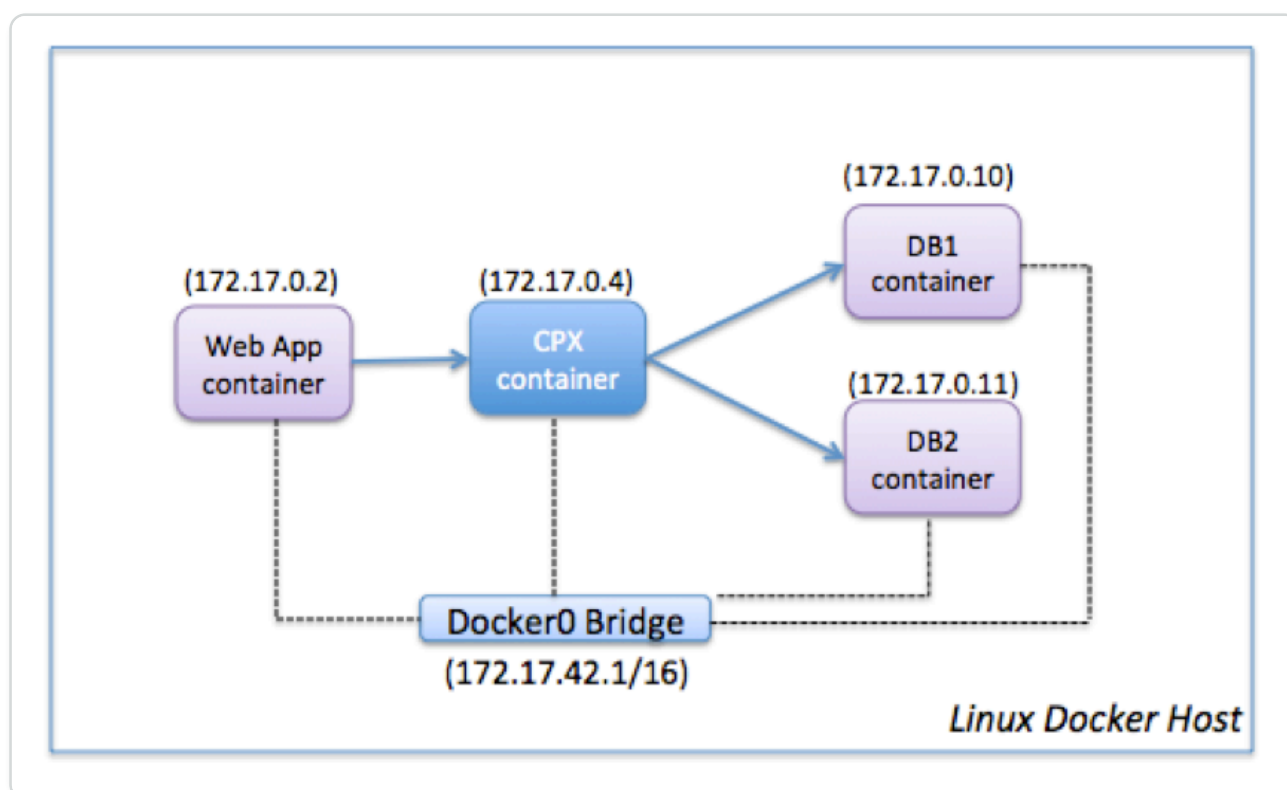
Deploy {{page.netscaler-cpx-short}} in a Single Host Network

In a single host network, the {{page.netscaler-cpx-short}} instance acts as a proxy between application containers on the same host. In this capacity, the {{page.netscaler-cpx-short}} instance provides scalability and security to the container-based applications. Additionally, it optimizes performance and also provides an insight into telemetry data.

In a single host network, the client, the servers, and the {{page.netscaler-cpx-short}} instance are deployed as containers on the same Docker host. All the containers are connected through docker0 bridge.

In this environment, the {{page.netscaler-cpx-short}} instance acts as a proxy for the applications provisioned as containers on the same Docker host.

The following figure illustrates the single host topology.



In this example, a web app container (172.17.0.2) is the client and the two database containers, DB1 (172.17.0.10) and DB2 (172.17.0.11), are the servers. The {{page.netscaler-cpx-short}} container (172.17.0.4) sits between the client and the servers acting as a proxy.

To enable the web application to communicate with the database containers through {{page.netscaler-cpx-short}}, you have to first configure two services on the {{page.netscaler-cpx-short}} container to represent the two servers. Then, configure a virtual server by using the {{page.netscaler-cpx-short}} IP address and a non-standard HTTP port (such as 81) because the {{page.netscaler-cpx-short}} reserves the standard HTTP port 80 for NITRO communication.

In this topology, you do not have to configure any NAT rules because the client and the server are on the same network.

To configure this scenario, run the following commands either by using the Jobs feature in NetScaler® MAS or by using NITRO APIs:

```
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
```

Deploy {{page.netscaler-cpx-short}} in a Multi-Host Network

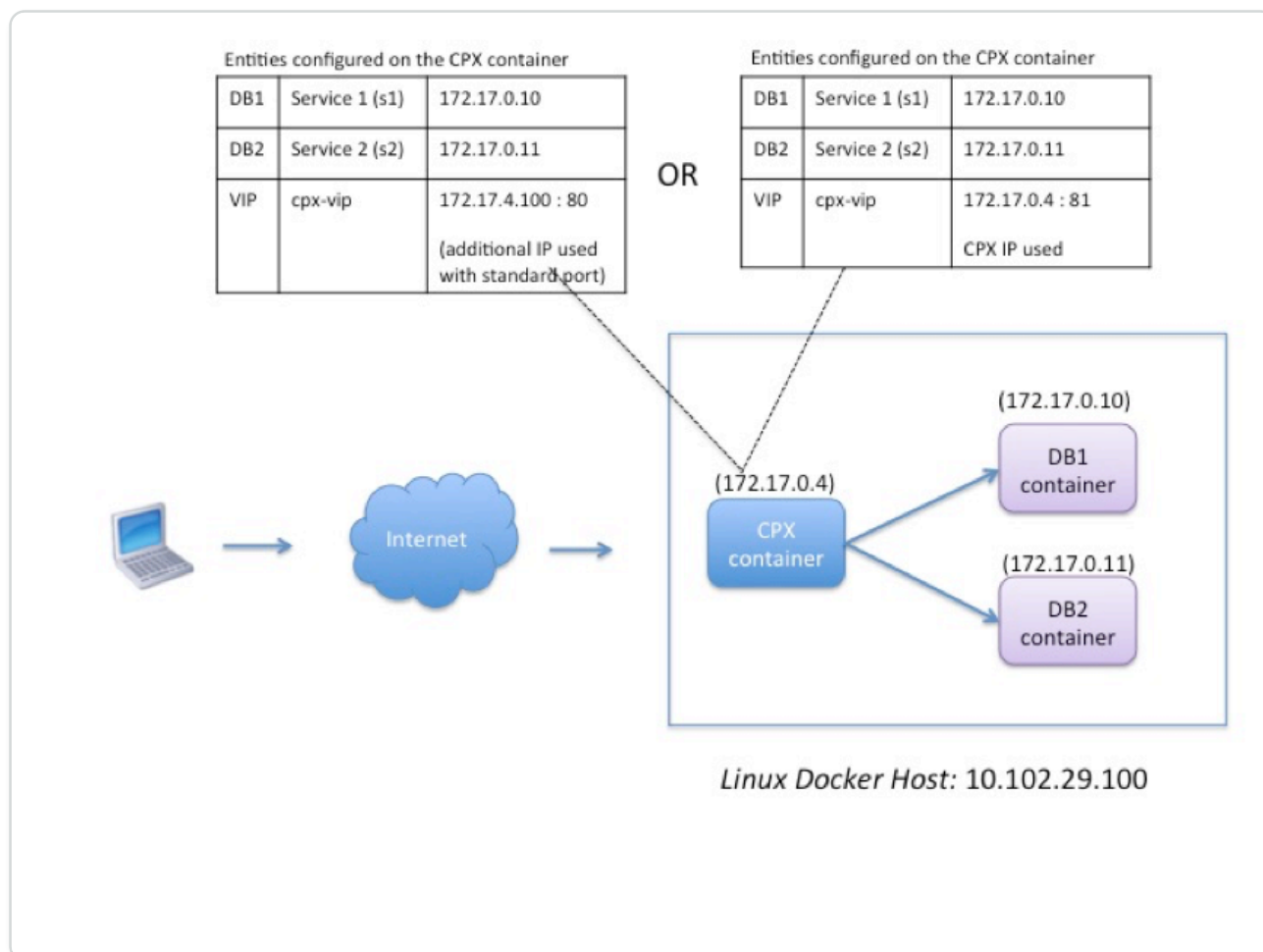
A {{page.netscaler-cpx-short}} instance in a multi-host network may be configured in a production deployment in the datacenter where it provides load balancing functions. It can further provide monitoring functions and analytics data.

In a multi-host network, the {{page.netscaler-cpx-short}} instances, backend servers, and the clients are deployed on different hosts. You can use multi-host topologies in production deployments where the {{page.netscaler-cpx-short}} instance load balances a set of container-based applications and servers or even physical servers.

Topology 1: {{page.netscaler-cpx-short}} and Backend Servers on Same Host; Client on a Different Network

In this topology, the {{page.netscaler-cpx-short}} instance and the database servers are provisioned on the same Docker host, but the client traffic originates from elsewhere on the network. This topology might be used in a production deployment where the {{page.netscaler-cpx-short}} instance load balances a set of container-based applications or servers.

The following diagram illustrates this topology.



In this example, the `{{page.netscaler-cpx-short}}` instance (172.17.0.4) and the two servers, DB1 (172.17.0.10) and DB2 (172.17.0.11) are provisioned on the same Docker host with IP address 10.102.29.100. The client resides elsewhere on the network.

The client requests originating from the Internet are received on the VIP configured on the `{{page.netscaler-cpx-short}}` instance, which then distributes the requests across the two servers.

There are two methods you can use to configure this topology:

Method 1: Using an additional IP address and standard port for the VIP

1. Configure the VIP on the `{{page.netscaler-cpx-short}}` container by using an additional IP address.
2. Configure an additional IP address for the Docker host.
3. Configure NAT rules to forward all traffic received on the Docker host's additional IP address to the VIP's additional IP address.
4. Configure the two servers as services on the `{{page.netscaler-cpx-short}}` instance.
5. Finally, bind the services to the VIP.

Note that in this example configuration, the 10.x.x.x network denotes a public network.

To configure this example scenario, run the following commands either by using the Jobs feature in NetScaler® MAS or by using NITRO APIs:

```
add service s1 172.17.0.10 HTTP 80

add service s2 172.17.0.11 HTTP 80

add lb vserver cpx-vip HTTP 172.17.4.100 80

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

Configure an additional public IP address for the Docker host and a NAT rule by running the following commands at the Linux shell prompt:

```
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
```

Method 2: Using the {{page.netscaler-cpx-short}} IP address for the VIP and by configuring port mapping:

1. Configure the VIP and the two services on the {{page.netscaler-cpx-short}} instance. Use a non-standard port, 81, with the VIP.
2. Bind the services to the VIP.
3. Configure a NAT rule to forward all traffic received on port 50000 of the Docker host to the VIP and port 81.

To configure this example scenario, run the following command at the Linux shell prompt while creating the {{page.netscaler-cpx-short}} container on all three Docker hosts:

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

After the {{page.netscaler-cpx-short}} instance is provisioned, run the following commands either by using the Jobs feature in NetScaler MAS or by using NITRO APIs:

```
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
```

NOTE:

If you have not configured port mapping during provisioning of the {{page.netscaler-cpx-short}} instance, then configure a NAT rule by running the following commands at the Linux shell prompt:

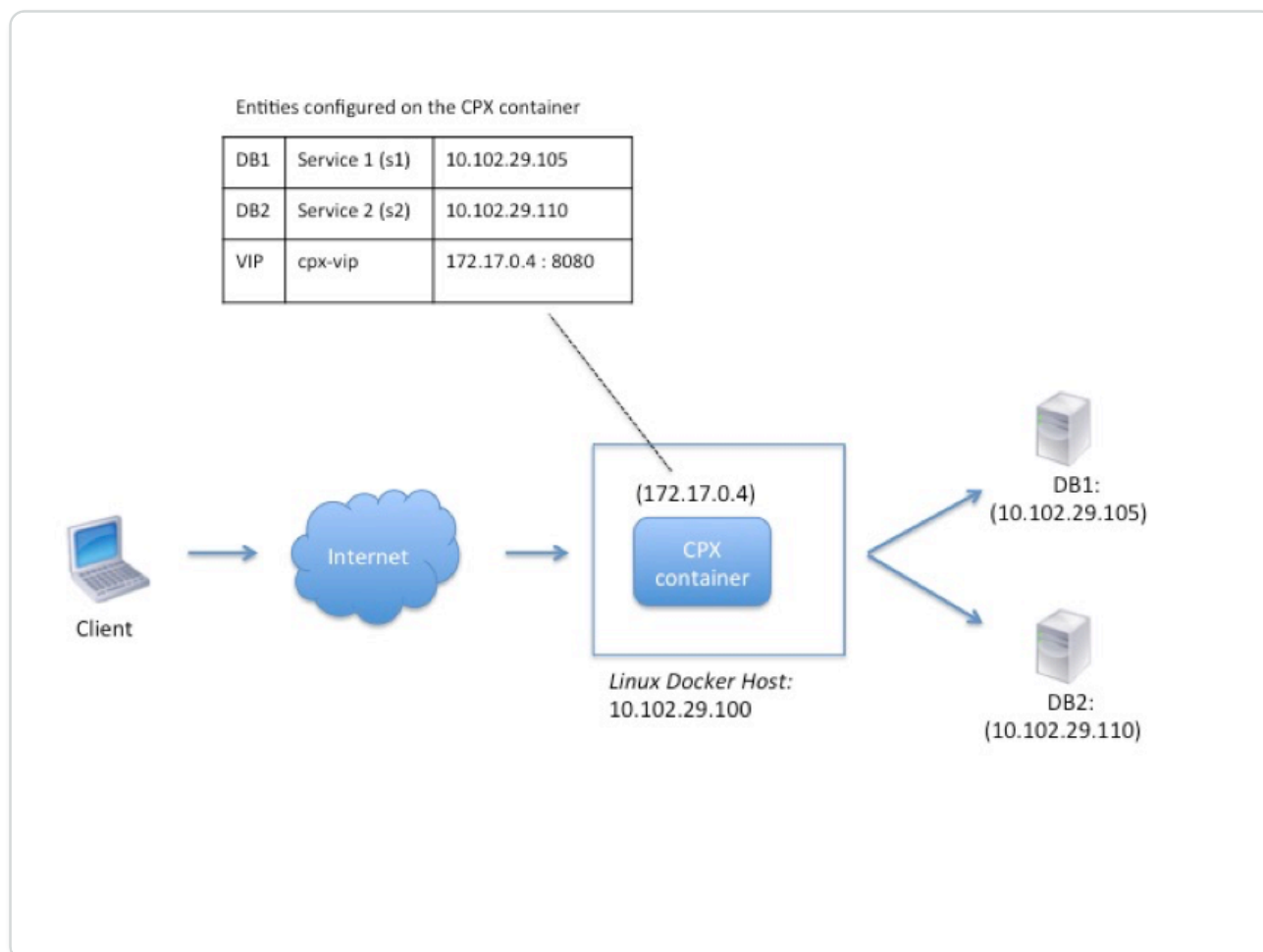
```
iptables -t nat -A PREROUTING -p tcp -m addrtype --dst-type LOCAL -m tcp --dport 50000 -j
DNAT --to-destination 172.17.0.4:81
```

Topology 2: {{page.netscaler-cpx-short}} with Physical Servers and Client

In this topology, only the {{page.netscaler-cpx-short}} instance is provisioned on a Docker host. The client and the servers are not container-based and reside elsewhere on the network.

In this environment, you can configure the {{page.netscaler-cpx-short}} instance to load balance traffic across the physical servers.

The following figure illustrates this topology.



In this example, the `page.netscaler-cpx-short` container (172.17.0.4) sits between the client and the physical servers acting as a proxy. The servers, DB1 (10.102.29.105) and DB2 (10.102.29.110), reside outside a Docker host on the network. The client request originates from the Internet and is received on the `page.netscaler-cpx-short`, which distributes it across the two servers.

To enable this communication between the client and the servers through `page.netscaler-cpx-short`, you have to first configure port mapping while creating the `page.netscaler-cpx-short` container. Then, configure the two services on the `page.netscaler-cpx-short` container to represent the two servers. And finally, configure a virtual server by using the `page.netscaler-cpx-short` IP address and the non-standard mapped HTTP port 8080.

Note that in the example configuration, the 10.x.x.x network denotes a public network.

To configure this example scenario, run the following command at the Linux shell prompt while creating the `page.netscaler-cpx-short` container:

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

Then, run the following commands either by using the Jobs feature in NetScaler MAS or by using NITRO APIs:

```
add service s1 HTTP 10.102.29.105 80

add service s2 HTTP 10.102.29.110 80

add lb vserver cpx-vip HTTP 172.17.0.4 8080

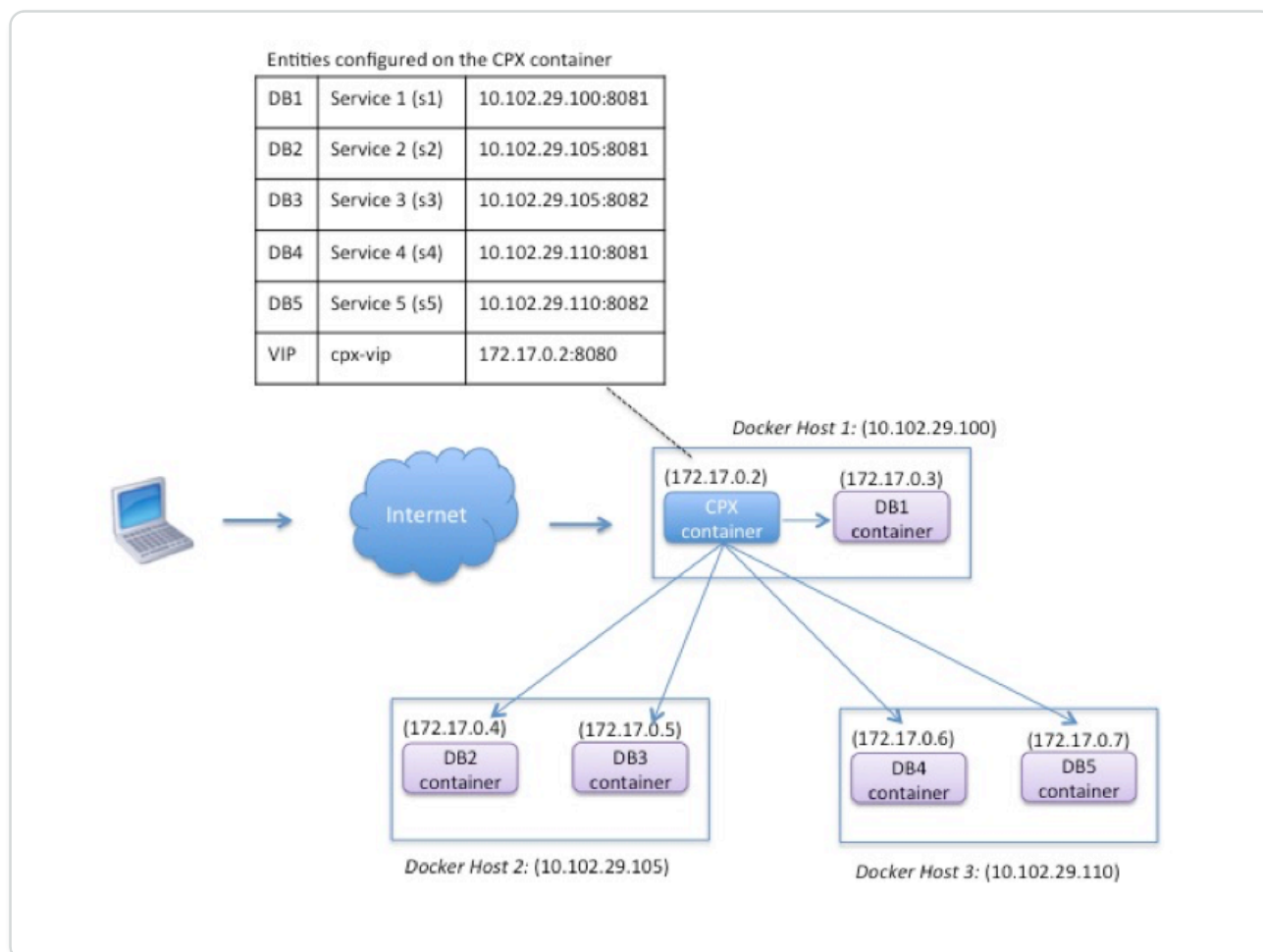
bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

Topology 3: {{page.netscaler-cpx-short}} and Servers Provisioned on Different Hosts

In this topology, the {{page.netscaler-cpx-short}} instance and the database servers are provisioned in different Docker hosts, and the client traffic originates from the Internet. This topology might be used in a production deployment where the {{page.netscaler-cpx-short}} instance load balances a set of container-based applications or servers.

The following diagram illustrates this topology.



In this example, the `page.netscaler-cpx-short` instance and a server (DB1) are provisioned on the same Docker host with IP address 10.102.29.100. Four other servers (DB2, DB3, DB4, and DB5) are provisioned on two different Docker hosts, 10.102.29.105 and 10.102.29.110.

The client requests originating from the Internet are received on the `page.netscaler-cpx-short` instance, which then distributes the requests across the five servers. To enable this communication, you must configure the following:

1. Set port mapping while creating your `page.netscaler-cpx-short` container. In this example, this means that you have to forward port 8080 on the container to port 8080 on the host. When the client request arrives on port 8080 of the host, it maps to port 8080 of the CPX container.
2. Configure the five servers as services on the `page.netscaler-cpx-short` instance. You have to use a combination of the respective Docker host IP address and mapped port to set these services.
3. Configure a VIP on the `page.netscaler-cpx-short` instance to receive the client request. This VIP should be represented by the `page.netscaler-cpx-short` IP address and port 8080 that was mapped to port 8080 of the host.
4. Finally, bind the services to the VIP.

Note that in the example configuration, the 10.x.x.x network denotes a public network.

To configure this example scenario, run the following command at the Linux shell prompt while creating the {{page.netscaler-cpx-short}} container:

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

Run the following commands either by using the Jobs feature in NetScaler MAS or by using NITRO APIs:

```
add service s1 10.102.29.100 HTTP 8081

add service s2 10.102.29.105 HTTP 8081

add service s3 10.102.29.105 HTTP 8082

add service s4 10.102.29.110 HTTP 8081

add service s5 10.102.29.110 HTTP 8082

add lb vserver cpx-vip HTTP 172.17.0.2 8080

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2

bind lb vserver cpx-vip s3

bind lb vserver cpx-vip s4

bind lb vserver cpx-vip s5
```

Deploy {{page.netscaler-cpx-short}} with direct access to the network

In bridge networking mode, you can configure {{page.netscaler-cpx-short}} instance to have direct access to the network. In this scenario, the incoming traffic is directly received on the {{page.netscaler-cpx-short}} virtual server IP (VIP).

To enable this communication, you have to first configure a public IP address on docker0 bridge. Then, remove the public IP address from the network port eth0 and bind the network port to the docker0 bridge.

Configure load balancing by adding the two services and then configure a network public IP address as the VIP on the {{page.netscaler-cpx-short}} instance. The client requests are received directly on the VIP.

In the example configuration, the 10.x.x.x network denotes a public network.

To configure this scenario, run the following command at the Linux shell prompt:

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

ip addr del 10.102.29.100/24 dev eth0;

brctl addif docker0 eth0;

ip route del default;

ip route add default via 10.102.29.1 dev docker0
```

Either by using the Jobs feature in NetScaler® MAS or by using NITRO APIs, run the following commands:

```
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
```

Configure {{page.netscaler-cpx-short}} in Kubernetes Using ConfigMaps

In Kubernetes, you can configure the {{page.netscaler-cpx-short}} instance using ConfigMaps. Using ConfigMaps you can dynamically configure the {{page.netscaler-cpx-short}} instance during instance startup.

Create a `cpx.conf` configuration file that includes NetScaler-specific configuration and bash shell commands that you want to run dynamically on the {{page.netscaler-cpx-short}} instance. The configuration file structure requires two types of tags, `#NetScaler Commands` and `#Shell Commands`. Under the `#NetScaler Commands` tag, you must add all the NetScaler commands to configure NetScaler-specific configuration on {{page.netscaler-cpx-short}} instance. Under the `#Shell Commands` tag, you must add the shell commands that you want to run on the {{page.netscaler-cpx-short}} instance.

IMPORTANT:

- The tags can be repeated multiple times in the configuration file.
- The configuration file can also include comments. Add a “#” character before comments.
- The tags are not case-sensitive.
- If there are failure scenarios while deploying the {{page.netscaler-cpx-short}} container with the configuration file, the failures are logged in the `ns.log` file.
- After the {{page.netscaler-cpx-short}} instance starts, if you change the ConfigMap, the updated configuration is applied only when the {{page.netscaler-cpx-short}} instance is restarted.

The following is a sample configuration file:

```
#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

```

Once you have created the configuration file, you must create a ConfigMap from the configuration file using the `kubectl create configmap` command.

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

In the example above, you can create a ConfigMap, `cpx-config` based on the configuration file `cpx.conf`. You can then use this ConfigMap in the YAML file used to deploy the `{{page.netscaler-cpx-short}}` instance.

You can view the created ConfigMap using the `kubectl get configmap` command.

```
root@node1:~/yaml# kubectl get configmap cpx-config -o yaml
```

Sample:

```

apiVersion: v1

data:

  cpx.conf: |

    #NetScaler Commands

```

```
    add lb vserver v1 http 1.1.1.1 80

    add service s1 2.2.2.2 http 80

    bind lb vserver v1 s1

#Shell Commands

    touch /etc/a.txt

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

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

    ls >> /etc/a.txt

#NetScaler Commands

    add lb vserver v2 http

#Shell Commands

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

#NetScaler Commands

    add lb vserver v3 http

#end of file

kind: ConfigMap

metadata:

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

    name: cpx-config

    namespace: default

    resourceVersion: "8865149"

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

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

You can specify the created ConfigMap, `cpx-config` in the YAML file used to deploy the {{page.netscaler-cpx-short}} instance as shown below:

```
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:12.1-51.16"

    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:

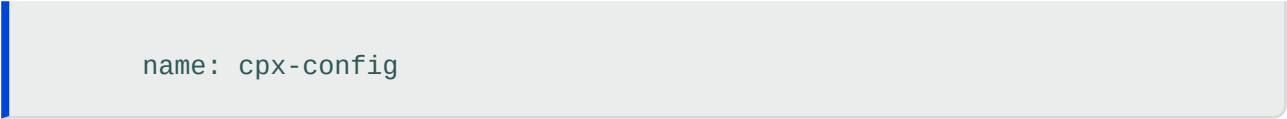
fieldPath: metadata.name

imagePullPolicy: Never

volumes:

- name: config-volume

configMap:
```



```
name: cpx-config
```

Once the {{page.netscaler-cpx-short}} instance is deployed and starts the configuration specified in the ConfigMap, `cpx-config` is applied to the {{page.netscaler-cpx-short}} instance.

Deploy {{page.netscaler-cpx-short}} Proxy on Google Compute Engine

This deployment guide describes how you can deploy {{page.netscaler-cpx-short}} with Docker on Google Cloud's Google Compute Engine (GCE) with NetScaler MAS running within the enterprise network. In this deployment, {{page.netscaler-cpx-short}} installed on GCE load balances two back-end servers, and NetScaler MAS provides licensing and analytics solutions.

{{page.netscaler-cpx-short}} is a container-based proxy that supports full Layer 7 functionality, SSL offload, multiple protocols, and NITRO API. NetScaler MAS provides management, licensing, and analytics solutions. As a licensing server, NetScaler MAS provides entitlement to {{page.netscaler-cpx-short}} instances that run on premises or in the cloud.

CPX and CPX Express are the same images. When licensed by NetScaler® MAS and installed, the CPX image in the Docker App Store (release 11 or 12) becomes a full CPX instance. Without a license, the CPX image becomes a CPX Express instance supporting 20 Mbps and 250 SSL connections.

Prerequisites

- 2 GB of memory and 1 vCPU dedicated to {{page.netscaler-cpx-short}}
- Docker open source available from GCE
- NetScaler MAS running on premises with internet or VPN connection to GCE

NOTE

For information about how to deploy NetScaler MAS see [Deploying NetScaler MAS](#).

Configuration Steps

You have to perform the following steps to configure this deployment.

1. Install Docker on a GCE VM.
2. Configure Remote API Communication with the Docker Instance.
3. Install {{page.netscaler-cpx-short}} Image.
4. Create a CPX Instance.
5. License {{page.netscaler-cpx-short}} Through NetScaler MAS.
6. Configure Load Balancing Services on {{page.netscaler-cpx-short}} and Verify the Configuration.
 - a. Install NGINX web servers.
 - b. Configure CPX for load balancing and verify distribution of load to both web services.

Step 1: Install Docker on a GCE VM

From GCE, create a Linux Ubuntu VM. Then, install Docker on the VM by using the commands shown in the following example:

```
$ 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/CakbNSq0b9o
ddfzRvGVeJVERt/Q/mlvEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
mYdoe/BJ/J/6rH0jq70mk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyu/jPpWFK28TR8yfVlzYFwibj5WK
dHM7ZTq1A1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccy0i+ULU0z8mHUJ3iEMIrpt
X+80KaN/ZjibfsBOCjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
H0oyC6qVe0gKrRiKxzymj0FIMLru/iFF5pSwcQB7PYlt8J0G80lAcPr6VCiN+4c
NKv03SdvA69dC0j79Pu09IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHHM1fDeCqkZ
hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Zaltn0svoGs+wx3Z/H6/8P5anog43U
65c0A+64Jj00rNDR8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+cOMHUU4HA7iM
zHrouAdYeTZeq0A7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFncTKdv+rwARAQAB
tDdEb2NrZXIgaUmVsZWZzSBUB29sIChyZWx1YXNlZG9ja2VyKSA8ZG9ja2VyQGRv
Y2t1ci5jb20+iQIcBBABCGAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4Syo0CV
cnybl18vzwBEcp4VCRbXvHvOXty1gccVIV8/aJqNKgBV97lY3vrp0yiIeB8ETQeg
srxFE7t/Gz0rsL0bqfLEHdmn5iBJRkhLfCpzje0nyB3Z0IJB6Uog0/msQVYe5CXJ
16uwr0AmoiCBLrVlDAktXvH9RWch0l0KZR2FpHu8h+uM0/zySqIidlyfLa3y5oH
scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdCAHR+bXhA/RW8QHeeMFliWw
7Z2jYHyuHmDnWG2yUrnCqAJTrwV+OfKRIZZJFBs4e88ru5h2ZIXdRepw/+COYj34
LyzxR2cxr2u/xvxwXCkSMe7F4KZAphD+1ws61FhnUMi/PERMYFTFuvPrCkq4gyBj
t3fFpZ2NR/fKw87Q0eVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsrxrkFQ2jNp
D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaH0
W1zb1Wy/CFr6VF1963jrjJgag0G6tNtBZLrc1ZgWh0QpeZZ5Lbvz2ZA5CqRrFAVc

```
wPNW1f0bFIRtqV6vuVluF0PCMAAnOnqR02w9t17iVQj03oVN0mbQi9vjuExXh1Yo
```

```
ScVeti06LSm1QfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBlAAoJEFyzYeVS
```

```
\+w0QeH0QAI6btAfYwYPuAjfRUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+1/R08nu0d
```

```
sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
```

```
NydGFzmeOKbAlrxIMECnSTG/TkFVY09Nt1v9vSN2BupmTagTRERxLZKnVswRzp+X
```

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

```
OK
```

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

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

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

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

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

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

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

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

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

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

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

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

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

Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main Sources
\[903 kB\]

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

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

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

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

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

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

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

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

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

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

```
Get:21 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe Sources \[2,468 B\  
  
Get:22 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
main Sources \[2,480 B\  
  
Get:23 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
main amd64 Packages \[3,500 B\  
  
Get:24 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe amd64 Packages \[3,820 B\  
  
Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe Translation-en \[1,592 B\  
  
Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64 Packages  
\[2,480 B\  
  
Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources  
\[47.7 kB\  
  
Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64  
Packages \[2,453 B\  
  
Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe Sources  
\[20.7 kB\  
  
Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse  
Sources \[1,140 B\  
  
Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted  
Sources \[2,292 B\  
  
Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64  
Packages \[150 kB\  
  
Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main  
Translation-en \[68.0 kB\  
  
Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe amd64  
Packages \[77.2 kB\  
  
Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe
```

```
Translation-en \[47.3 kB\]

Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse amd64
Packages \[2,832 B\]

Fetched 10.8 MB in 2s (4,206 kB/s)

Reading package lists... Done

Reading package lists...

Building dependency tree...

Reading state information...

The following additional packages will be installed:

aufs-tools cgroupfs-mount libltdl7

The following NEW packages will be installed:

aufs-tools cgroupfs-mount docker-engine libltdl7

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

Need to get 21.2 MB of archives.

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

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

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

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

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

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

Selecting previously unselected package aufs-tools.
```

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

Preparing to unpack .../aufs-
tools\_1%3a3.2+20130722-1.1ubuntu1\_amd64.deb ...

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

Selecting previously unselected package cgroupfs-mount.
Preparing to unpack .../cgroupfs-mount\_1.3\_all.deb ...

Unpacking cgroupfs-mount (1.3) ...

Selecting previously unselected package libltdl7:amd64.
Preparing to unpack .../libltdl7\_2.4.6-1\_amd64.deb ...

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

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

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

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

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

Setting up cgroupfs-mount (1.3) ...

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

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

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

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

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

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

```
→ /lib/systemd/system/docker.service.
```

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

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

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

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

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

```
Client:
```

```
Version: 17.05.0-ce
```

```
API version: 1.29
```

```
Go version: go1.7.5
```

```
Git commit: 89658be
```

```
Built: Thu May 4 22:15:36 2017
```

```
OS/Arch: linux/amd64
```

```
Server:
```

```
Version: 17.05.0-ce
```

```
API version: 1.29 (minimum version 1.12)
```

```
Go version: go1.7.5
```

```
Git commit: 89658be
```

```
Built: Thu May 4 22:15:36 2017
```

```
OS/Arch: linux/amd64
```

```
Experimental: false
```

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

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

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

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

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

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

for more information.

\$

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

Containers: 0

Running: 0

Paused: 0

```
Stopped: 0

Images: 0

Server Version: 17.05.0-ce

Storage Driver: aufs

Root Dir: /var/lib/docker/aufs

Backing Filesystem: extfs

Dirs: 0

Dirperm1 Supported: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9048e5e50717ea4497b757314bad98ea3763c145

runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228

init version: 949e6fa

Security Options:

apparmor
```

```
seccomp

Profile: default

Kernel Version: 4.8.0-51-generic

Operating System: Ubuntu 16.10

OSType: linux

Architecture: x86\_64

CPUs: 1

Total Memory: 3.613GiB

Name: docker-7

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

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

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

Experimental: false

Insecure Registries:

127.0.0.0/8

Live Restore Enabled: false

WARNING: No swap limit support

$
```

```
**$ sudo docker images**
```

```
REPOSITORY TAG IMAGE ID CREATED SIZE
```

```
$
```

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

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

```
$
```

Step 2: Configure Remote API Communication with the Docker Instance

Open port 4243 for API communication with the Docker instance. This is required for NetScaler MAS to communicate with the Docker instance.

```
$
```

```
$ **cd /etc/systemd/system**
```

```
$ **sudo vi docker-tcp.socket**
```

```
$
```

```
$ **cat docker-tcp.socket**
```

```
\[Unit\]
```

```
**Description=Docker Socket for the API
```

```
\[Socket\]

ListenStream=4243

BindIPv6only=both

Service=docker.service

\[Install\]

WantedBy=sockets.target**

$

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

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

$ **sudo systemctl enable docker.socket**

$ **sudo systemctl stop docker**

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

$ **sudo systemctl start docker**

$ **sudo systemctl status docker**

• docker.service - Docker Application Container Engine

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

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

Docs: https://docs.docker.com

Main PID: 4133 (dockerd)

Tasks: 16 (limit: 4915)
```

Memory: 30.1M

CPU: 184ms

CGroup: /system.slice/docker.service

|─4133 /usr/bin/dockerd -H fd://

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

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

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

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

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

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

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

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

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

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

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

```
$
```

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

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

```
(external)$
```

Step 3: Install {{page.netscaler-cpx-short}} Image

Get the {{page.netscaler-cpx-short}} image from Docker App Store. Note that the CPX Express and CPX have the same image. When licensed by NetScaler MAS and installed, the image becomes a full CPX instance with 1 Gbps of performance. Without a license, the image becomes a CPX Express instance supporting 20 Mbps and 250 SSL connections.

```
$ **sudo docker pull store/citrix/netscalercpx:12.0-41.16**
```

```
12.0-41.16: Pulling from store/citrix/netscalercpx
```

```
4e1f679e8ab4: Pull complete
```

```
a3ed95caeb02: Pull complete
```

```
2931a926d44b: Pull complete
```

```
362cd40c5745: Pull complete
d10118725a7a: Pull complete
1e570419a7e5: Pull complete
d19e06114233: Pull complete
d3230f008ffd: Pull complete
22bdb10a70ec: Pull complete
1a5183d7324d: Pull complete
241868d4ebff: Pull complete
3f963e7ae2fc: Pull complete
fd254cf1ea7c: Pull complete
33689c749176: Pull complete
59c27bad28f5: Pull complete
588f5003e10f: Pull complete

Digest:
sha256:31a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe

Status: Downloaded newer image for store/citrix/netscalercpx:12.0-41.16

$

$ **sudo docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE
store/citrix/netscalercpx 12.0-41.16 6fa57c38803f 3 weeks ago 415MB
```

\$

Step 4: Create a CPX Instance

Install the CPX image on the Docker host. Open ports for specific services, as shown in the example below, and specify an IP address for NetScaler MAS:

```
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/netscalercpx:12.0-41.16**
```

```
44ca1c6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
```

```
$
```

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

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

```
44ca1c6c0907 store/citrix/netscalercpx:12.0-41.16 "/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

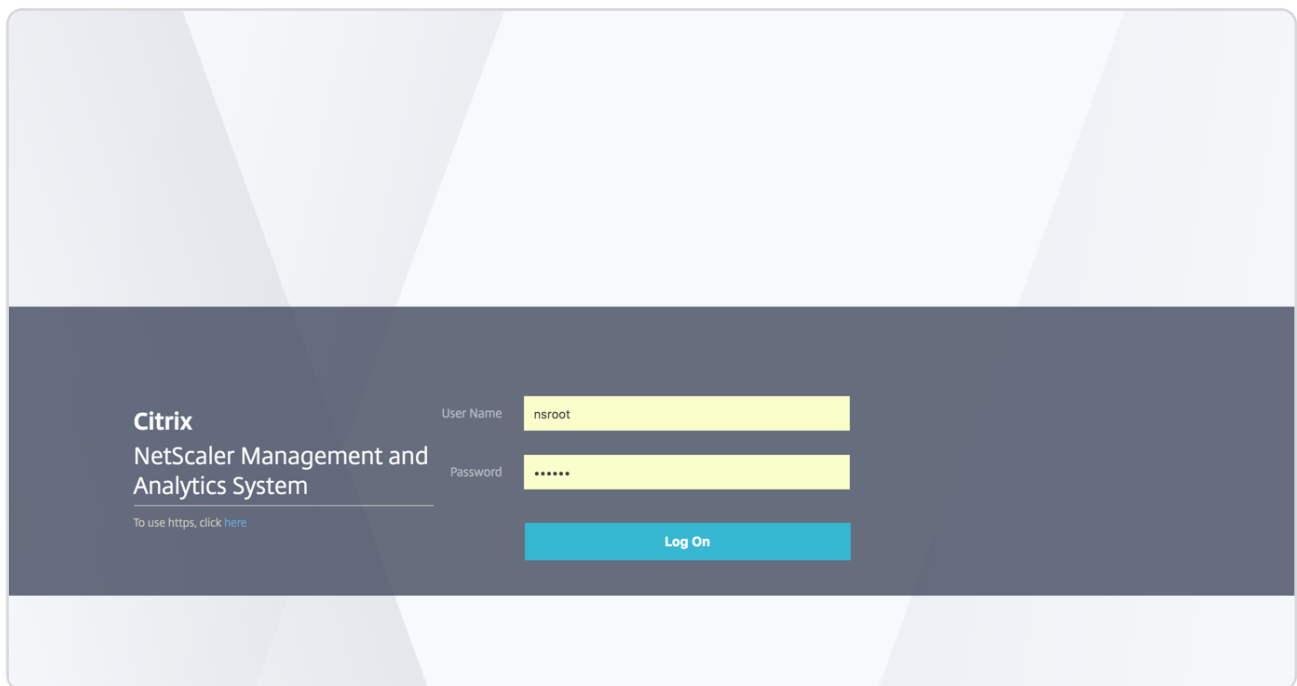
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80

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

\$

Step 5: License {{page.netscaler-cpx-short}} Through NetScaler MAS

Assuming NetScaler MAS is running on premises, you should be able to validate that CPX is communicating with MAS and sending information. The following images show CPX retrieving a license from NetScaler MAS.



Citrix NetScaler Management and Analytics System

May 31 2017 13:14:20 GMT

nsroot

Search here

Applications

Networks

Dashboard

Instances

Instance Groups

Licenses

System Licenses

Third Party Licenses

CPX Licenses

Events

SSL Dashboard

Configuration Jobs

Configuration Audit

Sites

Network Functions

Network Reporting

Analytics

Orchestration

System

Downloads

Networks > License Settings

License Server Port Settings

Proxy Server Port
0

License Server Port
27000

Vendor Daemon Port
7279

License Files

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

Add License File Apply Licenses Delete Download

☐	Name	Last Modified	Size
☐	FID__3bf0b423_15ba7640cc6_2664.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__3bf0b423_15ba7640cc6_2672.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__2a2386a8_15b93284902_487e.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__2a2386a8_15b93284902_4878.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__3bf0b423_15ba7640cc6_5281.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__2a2386a8_15b93284902_4870.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__3bf0b423_15ba7640cc6_527b.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__2a2386a8_15b93284902_486a.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__3bf0b423_15ba7640cc6_5275.lic	2017-05-23 21:30:15	1.10 KB
☐	FID__2a2386a8_15b93284902_4864.lic	2017-05-23 21:30:15	1.10 KB

License Expiry Information

Feature	Count	Days To Expiry
No items		

Citrix NetScaler Management and Analytics System

Jun 05 2017 15:09:41 GMT

nsroot

Search here

Applications

Networks

Dashboard

Instances

Instance Groups

Licenses

System Licenses

Third Party Licenses

CPX Licenses

Events

SSL Dashboard

Configuration Jobs

Configuration Audit

Sites

Network Functions

Network Reporting

Analytics

Orchestration

System

Downloads

Networks > Instances > NetScaler CPX

NetScaler CPX

Instances 3 Docker Host 0

Add Provision Delete Dashboard Reboot Action

Search

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

Citrix NetScaler Management and Analytics System

Jun 05 2017 15:10:23 GMT

nsroot

Search here

Applications

Networks

Dashboard

Instances

Instance Groups

Licenses

System Licenses

Third Party Licenses

CPX Licenses

Events

SSL Dashboard

Configuration Jobs

Configuration Audit

Sites

Network Functions

Network Reporting

Analytics

Orchestration

System

Downloads

Networks > License Settings > CPX Licenses

CPX Licenses

Instances

10.0%

Total 10 Used 1

The following instances are consuming instance license.

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

© 1997–2026 Citrix Systems, Inc. All rights reserved.

95

Step 6: Configure Load Balancing Services on {{page.netscaler-cpx-short}} and Verify the Configuration

First, install NGINX web servers on the Docker host. Then, configure load balancing on {{page.netscaler-cpx-short}} to load balance the two web servers, and test the configuration.

Install NGINX Web Servers

Use the commands shown in the following example to install NGINX web servers.

```
$ sudo docker pull nginx

Using default tag: latest

latest: Pulling from library/nginx

Digest:
sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268

Status: Image is up to date for nginx:latest


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

098a77974818f451c052ecd172080a7d45e446239479d9213cd4ea6a3678616f


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

bbdac2920bb4085f70b588292697813e5975389dd546c0512daf45079798db65
```

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

Chain PREROUTING (policy ACCEPT)

target prot opt source destination

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


Chain INPUT (policy ACCEPT)

target prot opt source destination


Chain OUTPUT (policy ACCEPT)

target prot opt source destination

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


Chain POSTROUTING (policy ACCEPT)

target prot opt source destination

MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443

MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80

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

```
MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80

MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80


Chain DOCKER (2 references)

target prot opt source destination

RETURN all -- 0.0.0.0/0 0.0.0.0/0

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

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

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

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

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

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

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

$
```

Configure CPX for load balancing and verify distribution of load to both web services

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

root@localhost's password:

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

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

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

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

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

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

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

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

```
exec: add service web1 172.17.0.3 HTTP 80
```

```
Done
```

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

```
exec: add service web2 172.17.0.4 HTTP 80
```

```
Done
```

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

```
exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
```

```
Done
```

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

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

```
Done
```

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

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

Done

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

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

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

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

State: UP

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

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

Effective State: UP

Client Idle Timeout: 180 sec

Down state flush: ENABLED

Disable Primary Vserver On Down : DISABLED

Appflow logging: ENABLED

Port Rewrite : DISABLED

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

Configured Method: LEASTCONNECTION

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

Mode: IP

```
Persistence: NONE

Vserver IP and Port insertion: OFF

Push: DISABLED Push VServer:

Push Multi Clients: NO

Push Label Rule: none

L2Conn: OFF

Skip Persistency: None

Listen Policy: NONE

IcmpResponse: PASSIVE

RHstate: PASSIVE

New Service Startup Request Rate: 0 PER\_SECOND, Increment Interval: 0

Mac mode Retain Vlan: DISABLED

DBS\_LB: DISABLED

Process Local: DISABLED

Traffic Domain: 0

TROFS Persistence honored: ENABLED

Retain Connections on Cluster: NO


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

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

Done

root@44ca1c6c0907:~\#
```

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

\<!DOCTYPE html\>

\<html\>

\<head\>

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

\<style\>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

}

\</style\>

\</head\>

\<body\>

\<h1\>Welcome to nginx\!\</h1\>

\<p\>If you see this page, the nginx web server is successfully installed
and

working. Further configuration is required.\</p\>

\<p\>For online documentation and support please refer to

\<a href="http://nginx.org/"\>nginx.org\</a\>.\<br/\>
```

Commercial support is available at

nginx.com.

Thank you for using nginx.

(external)\$

![localized image](/content/dam/netscaler/us/en/cpx/12-1/media/cpx-image-5.png)

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

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

					Dload	Upload	Total	Spent	Left
Speed									

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

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

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1893	0	--:--:--	--:--:--	--:--:--
1894										

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

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1884	0	--:--:--	--:--:--	--:--:--
1883										

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

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1917	0	--:--:--	--:--:--	--:--:--
1924										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
---------	------------	---------	---------	-------	------	------	------

Current

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1877	0	--:--:--	--:--:--	--:--:--
1883										

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

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1852	0	--:--:--	--:--:--	--:--:--
1848										

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

						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1860	0	--:--:--	--:--:--	--:--:--
1865										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1887	0	--:--:--	--:--:--	--:--:--
1888										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1802	0	--:--:--	--:--:--	--:--:--
1800										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1902	0	--:--:--	--:--:--	--:--:--
1906										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1843	0	--:--:--	--:--:--	--:--:--
1848										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1862	0	--:--:--	--:~:~:~	--:~:~:~
1860										

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

```
100    612    100    612      0      0    1806      0  --:--:--  --:--:--  --:--:--
1810
```

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

```
                                Dload  Upload  Total  Spent  Left
Speed
```

```
100    612    100    612      0      0    1702      0  --:--:--  --:--:--  --:--:--
1704
```

(external)\$

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

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

Virtual Server Summary

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

inactSvcs

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

Virtual Server Statistics

	Rate (/s)	Total
Vserver hits	0	101

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

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

```
Current Multipath TCP subflows          --          0

Apdex for client response times.        --          1.00

Average client TTLB                     --          0

web1          172.17.0.3      80      HTTP      UP      51
0/s

web2          172.17.0.4      80      HTTP      UP      50
0/s

Done

root@44ca1c6c0907:~\#
```

{{page.netscaler-cpx-short}}

troubleshooting

This document explains how to troubleshoot issues that you may encounter while using {{page.netscaler-cpx-short}}. Using this document, you can collect logs to determine the causes and apply workarounds for some of the common issues related to the installation and configuration of {{page.netscaler-cpx-short}}.

- How can I view {{page.netscaler-cpx-short}} logs?

You can view {{page.netscaler-cpx-short}} logs using the `kubectl logs` command if {{page.netscaler-cpx-short}} is deployed with the `tty:true` option. You can run the following command to display the logs:

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

Example,

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

The following is an example of the {{page.netscaler-cpx-short}} pod deployment with the `tty:true` option:

```
containers:
  - name: cpx-ingress
    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-58.30"
    tty: true
    securityContext:
      privileged: true
    env:
```

You can find more boot logs in the `/cpx/log/boot.log` file of `{{page.netscaler-cpx-short}}` file system.

Note: To get the pod name, run the `kubectl get pods -o wide` command.

- How can I collect the technical support bundle from `{{page.netscaler-cpx-short}}`?

You can run the following command on the shell interface of the Kubernetes master node to collect the `{{page.netscaler-cpx-short}}` technical support bundle:

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

You can view the technical support bundle in the `/var/tmp/support` directory of the `{{page.netscaler-cpx-short}}`'s file system. Use `scp` or `kubectl cp` to copy the technical support bundle from `{{page.netscaler-cpx-short}}` to the desired destination.

Example:

```
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
```

- Why is `{{page.netscaler-cpx-short}}` pod stuck while booting?

You can check the pod status using the `kubectl describe pods` command. Run the following command to know the pod status:

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

Example:

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

If the pod events show that container is started, then you must check the pod logs.

- How do I copy files between the {{page.netscaler-cpx-short}} pod and the Kubernetes master node?

It is recommended to use the volume mount feature of docker to mount the `/cpx` directory to the file system of the host. If a {{page.netscaler-cpx-short}} container exits core-dumps, logs and other important data are available on the mount point.

You can use any one of the following commands to copy files between the {{page.netscaler-cpx-short}} pod and the Kubernetes master node:

kubectl cp: You can run the following command to copy files from pod to node:

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

Example:

```
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: You can use the command to copy files between the {{page.netscaler-cpx-short}} pod and the Kubernetes node. Run the following command to copy files from pod to node. When it prompts for the password, provide the password for the SSH user:

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

Example:

```
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:~#
```

- How do I capture packets on {{page.netscaler-cpx-short}}?

To capture packets on {{page.netscaler-cpx-short}}, launch the shell interface of {{page.netscaler-cpx-short}} using the `kubectl exec` command. Run the following command to launch the shell interface of the {{page.netscaler-cpx-short}} pod:

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

Example:

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

And, run the following command to begin packet capture:

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

If you want to stop the ongoing packet capture, run the following command:

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

You can view the packets captured in a `.cap` file in the `/cpx/nstrace/time-stamp` directory on the {{page.netscaler-cpx-short}} file system.

- Why is the license server not configured even when {{page.netscaler-cpx-short}} is deployed with the `LS_IP=<ADM-IP>` environment variable?

Ensure that the license server is accessible from the node on which {{page.netscaler-cpx-short}} is deployed. You can use the `ping <ADM-IP>` command to verify the connectivity from the {{page.netscaler-cpx-short}} node to Citrix ADM.

If Citrix ADM is accessible from the node, then you must verify the license server configuration logs in the `/cpx/log/boot.log` file. You can also check for license server configuration using the following command on the shell interface of the {{page.netscaler-cpx-short}} pod:

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

Example:

```
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
```

- Why is pooled license not configured on {{page.netscaler-cpx-short}} even after a successful license server configuration on {{page.netscaler-cpx-short}}?

Verify the license configuration logs in the `/cpx/log/boot.log` file. You can also verify the configured pooled license on {{page.netscaler-cpx-short}} using the following command on the shell interface of the {{page.netscaler-cpx-short}} pod:

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

Example,

```
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
```

Also, ensure that the required license files are uploaded in the license server. You can also verify the available licenses on the license server once it is successfully configured on `{{page.netscaler-cpx-short}}` by using the following command. Run the command on the shell interface of `{{page.netscaler-cpx-short}}` pod:

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

Example:

```
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

- Why do NITRO API calls get *Connection Refused* response from {{page.netscaler-cpx-short}}?

The default port for NITRO APIs is 9080 (unsecure) and 9443 (secure) from the {{page.netscaler-cpx-short}} release 12.1 onwards. Ensure that the NITRO port of {{page.netscaler-cpx-short}} you try to access is exposed on the pod. You can run the `kubectl describe` command to view the exposed and mapped port of the {{page.netscaler-cpx-short}} container in the {{page.netscaler-cpx-short}} container section:

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

Example:

```
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
```

- Why does the NSPPE process in {{page.netscaler-cpx-short}} consume most of the CPU usage even when there is no or little traffic?

If {{page.netscaler-cpx-short}} is deployed with the `CPX_CONFIG='{"YIELD": "NO"}'` environment variable, the NSPPE process consumes 100 percent CPU usage even when there is no or little traffic. If you want the NSPPE process not to consume the CPU usage, you must deploy {{page.netscaler-cpx-short}} without the `CPX_CONFIG='{"YIELD": "NO"}'` environment variable. By default, the NSPPE process in CPX is configured not to hog or consume the CPU usage.

- Why is {{page.netscaler-cpx-short}} not listed in Citrix ADM even when it was deployed with the required environment variables for registration with Citrix ADM?

You can find the logs for {{page.netscaler-cpx-short}} registration with Citrix ADM in the `/cpx/log/boot.log` file on the {{page.netscaler-cpx-short}} file system.

You can verify the accessibility of the Citrix ADM IP address from the {{page.netscaler-cpx-short}} pod using the `ping` command. Also, ensure that all the required environment variables for Citrix ADM registration are configured for the {{page.netscaler-cpx-short}} container.

- `NS_MGMT_SERVER= <ADM-IP>`
 - `NS_MGMT_FINGER_PRINT= <ADM-Fingerprint>`
 - `HOST= <Node-IP>`
 - `NS_HTTP_PORT= <Mapped-HTTP-Port-on-Node>`
 - `NS_HTTPS_PORT= <Mapped-HTTPS-Port-on-Node>`
 - `NS_SSH_PORT= <Mapped-SSH-Port-on-Node>`
 - `NS_SNMP_PORT= <Mapped-SNMP-Port-on-Node>`
 - `NS_ROUTABLE=0` ({{page.netscaler-cpx-short}} pod IP address is not routable from outside.)
- Why does `cli_script.sh` show *Invalid user name or password* error message after changing the password for nsroot user?

The command `cli_script.sh` is a wrapper utility for NSCLI on {{page.netscaler-cpx-short}}. It runs the first argument as command string or file path and the second argument is optional which is credentials. If the password for the nsroot user is changed, you need to provide credentials as the second argument to `cli_script.sh`. You can run the following command to run NSCLI with credentials:

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

Example:

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

exec: show ns ip
```

ERROR: Invalid username or password

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

exec: show ns ip

Ipaddress	Traffic Domain		Type	Mode	Arp
Icmp	Vserver	State			
-----	----		----	---	----
-----	-----	-----			
172.17.0.3	0		NetScaler IP	Active	Enabled
Enabled NA		Enabled			
192.0.0.1	0		SNIP	Active	Enabled
Enabled NA		Enabled			
Done					