



NetScaler CPX 14.1

Contents

About NetScaler CPX	2
Architecture and Traffic Flow	4
NetScaler CPX licensing	7
NetScaler CPX support for Kubernetes probes	13
Deploying a NetScaler CPX Instance in Docker	15
Adding NetScaler CPX Instances to NetScaler Console	22
NetScaler CPX License Aggregator	27
Configuring NetScaler CPX	32
Configuring AppFlow on a NetScaler CPX instance	36
Configuring NetScaler CPX Using a Configuration File	39
Dynamic Routing support in NetScaler CPX	40
Configuring Docker Logging Drivers	43
Upgrading a NetScaler CPX Instance	44
Using Wildcard Virtual Servers in NetScaler CPX Instance	46
Deploy NetScaler CPX as a Proxy to Enable East-West Traffic Flow	47
Deploy NetScaler CPX in a Single Host Network	51
Deploy NetScaler CPX in a Multi-Host Network	52
Deploy NetScaler CPX with direct access to the network	57
Configure NetScaler CPX in Kubernetes Using ConfigMaps	58
Custom metrics configuration for NetScaler CPX using ConfigMap	61
Deploy NetScaler CPXs as Local DNS Caches for Kubernetes Nodes	63
Deploy NetScaler CPX Proxy on Google Compute Engine	67
NetScaler CPX troubleshooting	87

About NetScaler CPX

September 7, 2026

NetScaler CPX is a container-based application delivery controller that can be provisioned on a Docker host. NetScaler CPX 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 NetScaler CPX instances as standalone instances on a Docker host.

A NetScaler CPX instance provides throughput of up to 1 Gbps.

As a containerized form factor of NetScaler, NetScaler CPX integrates well into the Kubernetes environment and forms an integral part of NetScaler cloud native solution. NetScaler cloud native solution helps you to create and deliver software applications with speed, agility, and efficiency in a Kubernetes environment. Using NetScaler cloud native solution, you can ensure enterprise grade reliability and security for your Kubernetes environment.

For more information, see [NetScaler cloud native solution](#).

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

NetScaler CPX supports the following features:

- Application availability
 - L4 load balancing and L7 content switching
 - SSL offloading
 - IPv6 protocol translation
 - Microsoft SQL, MySQL load balancing
 - AppExpert rate controls
 - Subscriber-aware traffic steering
 - Surge protection and priority queuing
 - Dynamic routing protocols
- Application acceleration
 - Client and server TCP optimizations
 - Cache redirection
 - AppCompress
 - AppCache
- Application security

- L7 rewrite and responder
- L4 DoS defenses
- L7 DoS defenses
- Web Application Firewall (WAF). NetScaler CPX supports all WAF features which are supported on other NetScaler form factors. For information about supported WAF features, see [NetScaler Web App Firewall](#).
- TCP protocol optimization
 - Multi-path TCP
 - Binary Increase Congestion Control (BIC) and cubic TCP
- Simple manageability
 - Web logging
 - AppFlow®
 - NetScaler® Application Delivery Management
 - Action analytics
- Application optimization
 - Integrated caching
- BGP Routing and Route Health Injection (RHI)

Note:

Interface features such as Rx, Tx, GRO, GSO, and LRO are disabled for interfaces (Linux host) allocated to the NetScaler CPX appliance. These features remain in the disabled state even after the NetScaler CPX appliance is stopped. Also, the MTU is changed to 1500 bytes for such interfaces.

Supported Platforms

NetScaler CPX is supported on the following platforms:

- Kubernetes
- Red Hat OpenShift
- Public clouds
 - Amazon Elastic Kubernetes Service (EKS)
 - Azure Kubernetes Service (AKS)
 - Google Kubernetes Engine (GKE)
- Rancher
- Pivotal Container Service (PKS)
- Docker version 1.12 and above

Architecture and Traffic Flow

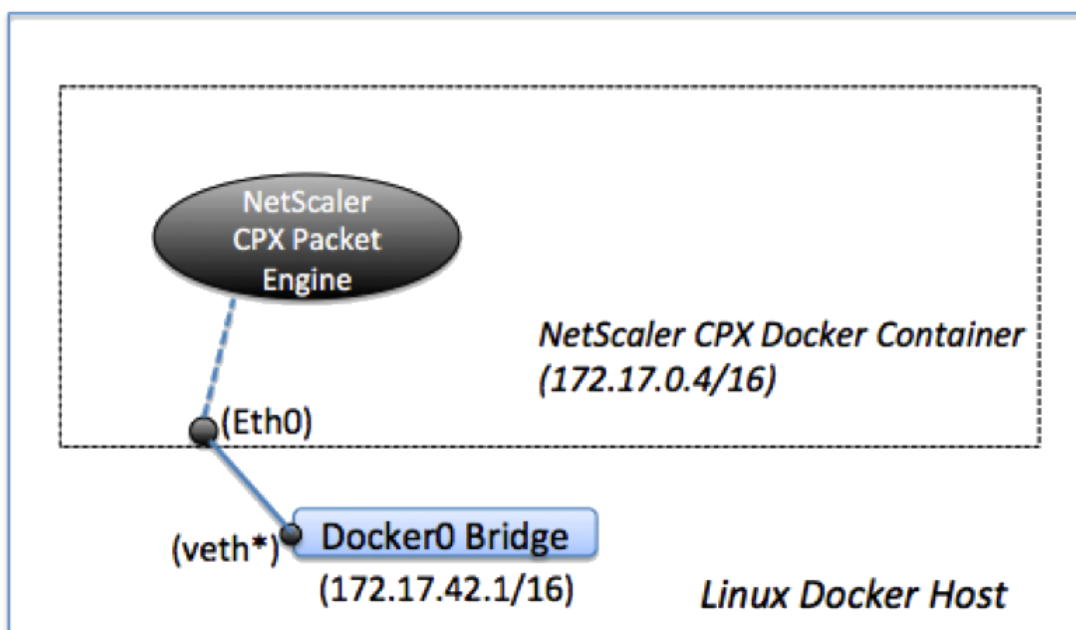
September 8, 2026

This section describes NetScaler CPX bridge mode architecture and traffic flow. NetScaler CPX can be deployed in host mode also.

When you provision a NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX instance is done through the Docker network. All incoming traffic received from the docker0 bridge is received by the eth0 interface on the NetScaler CPX instance and processed by the NetScaler CPX packet engine.

The following figure illustrates the architecture of a NetScaler CPX instance on a Docker host.



How Single IP Address Works on NetScaler CPX

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

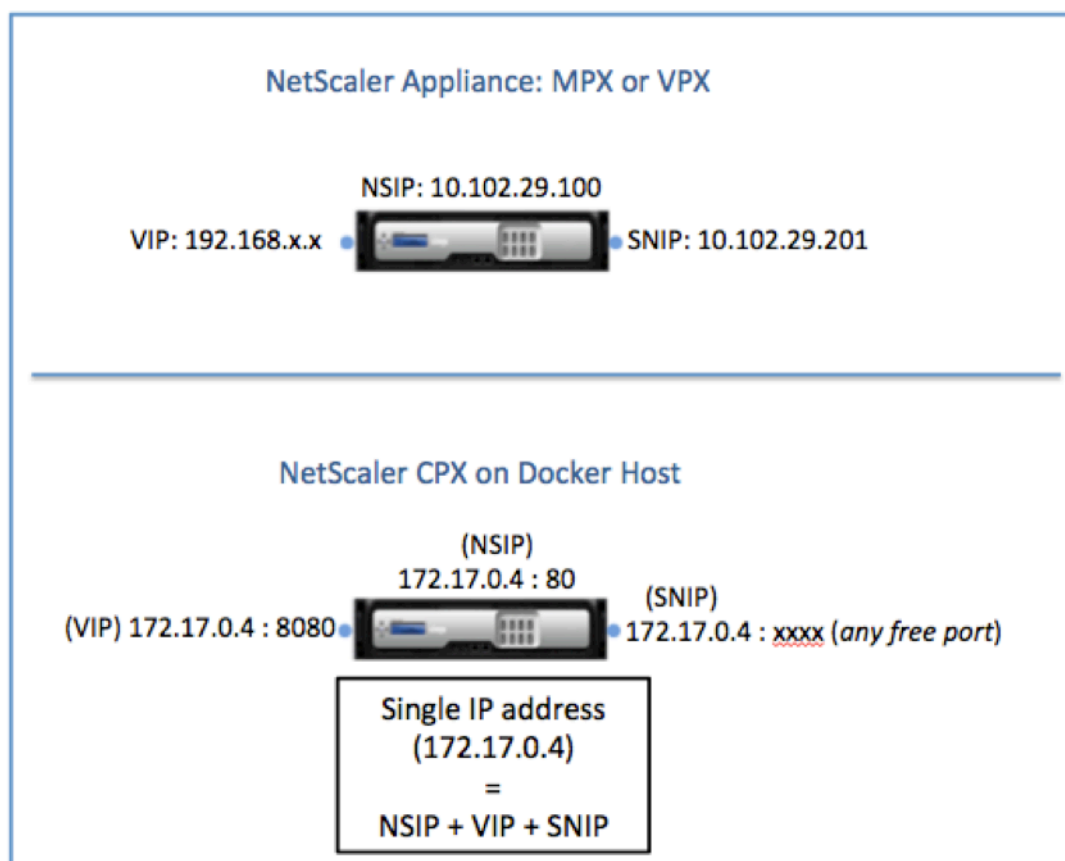
- Management IP address called the NetScaler IP (NSIP) address

- Subnet IP (SNIP) address for communicating with the server farm
- Virtual server IP (VIP) address(es) for accepting client requests

A NetScaler CPX 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 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 VIP(s).

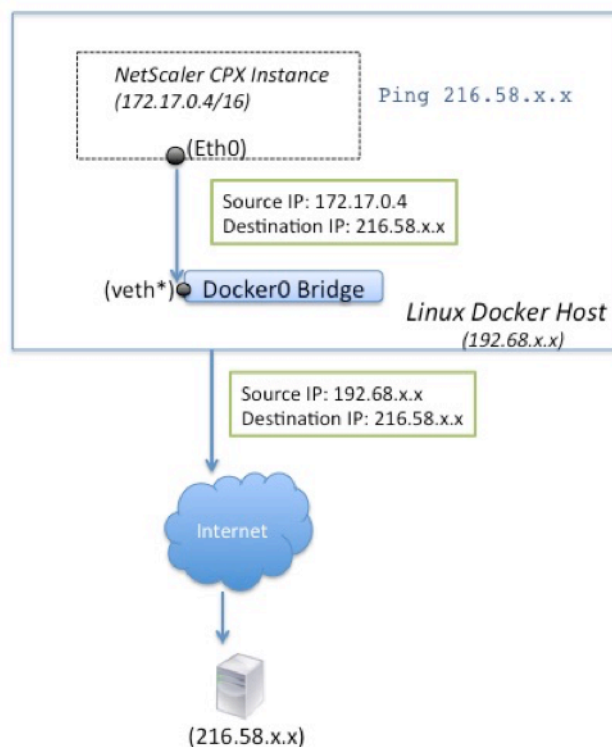
The following image illustrates how a single IP address is used to perform the functions of NSIP, SNIP, and VIP(s).



Traffic Flow for Requests Originating from the NetScaler CPX Instance

Docker implicitly configures IP tables and a NAT rule to direct traffic originating from the NetScaler CPX instance to the docker0 IP address.

The following figure illustrates how a ping request originating from a NetScaler CPX instance reaches the destination.



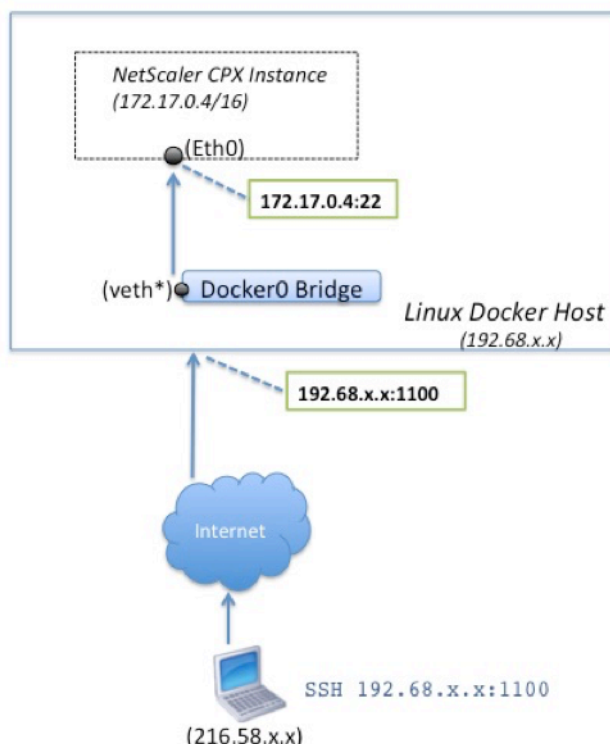
In this example, the ping request is sent by the packet engine on eth0 interface with source IP address as the NetScaler CPX IP address (172.17.0.4). The Docker host, then, performs network address translation (NAT) to add the host IP address (192.68.x.x) as the source IP address and sends the request 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 NetScaler CPX instance on the eth0 interface.

Traffic Flow for Requests Originating from the External Network

To enable external communication, while provisioning NetScaler CPX, 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX instance is 172.17.0.4. The SSH port 22 of NetScaler CPX 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.

NetScaler CPX licensing

September 7, 2026

NetScaler CPX is a container-based application delivery controller that can be provisioned on a Docker host to load balance microservice-based applications. You need a licensed NetScaler CPX for better performance of application delivery. NetScaler CPX supports legacy pool licensing and new [Citrix Platform or Universal Hybrid Multi-Cloud licensing](#). All NetScaler CPX instances are licensed through only NetScaler Console (formerly NetScaler ADM).

NetScaler Console is available in both on-premises and cloud service offerings. You can use NetScaler Console to manage pooled capacity or Universal Hybrid Multi-Cloud (UHMC) or Citrix Platform License (CPL) licenses of all NetScaler form factors.

For information about NetScaler Console on-premises, see [NetScaler Console on-premises](#). For information about NetScaler Console service, see [NetScaler Console service](#).

Types of NetScaler CPX licensing

The following types of NetScaler CPX licenses are supported:

- NetScaler CPX CPX Express - Free, Developer Centric NetScaler for Kubernetes
- Commercial NetScaler CPX –Production ready, Enterprise support proxy

NetScaler CPX Express

NetScaler CPX Express is a software edition that is free-of-cost for on-premises and cloud deployments. When you download a NetScaler CPX instance from the [Quay](#) repository, the NetScaler CPX Express license is applied to the NetScaler CPX instance by default. All the NetScaler CPX features are available with the NetScaler NetScaler CPX Express license. For information about the supported features on CPX, see [About NetScaler CPX](#). The NetScaler community support is available with NetScaler CPX Express but not the enterprise support.

The NetScaler CPX Express license offers 100 Mbps bandwidth capacity. You must license your NetScaler CPX instance for higher capacity, production deployment, and to avail enterprise support.

Note:

Starting from NetScaler CPX version 14.1 build 17.x, the bandwidth of NetScaler CPX Express license is increased from 20 Mbps to 100 Mbps. This increased bandwidth enables Kubernetes developers to try out NetScaler proxy.

Commercial NetScaler CPX

LAS will be the only way to activate and license NetScaler instances after April 15, 2026, supporting NetScaler Flexed licenses (CPL/UHMC), legacy NetScaler Bandwidth Pooled licenses. NetScaler CPX will support only NetScaler Flexed licenses (CPL/UMC) or legacy NetScaler pooled bandwidth licenses post LAS transition. For more information, see [transition to flexed licensing](#) for more details.

Supported pooled capacity for NetScaler CPX instances

Product	Maximum bandwidth	Minimum bandwidth	Minimum instances	Maximum instances	Minimum bandwidth unit
NetScaler CPX	40000 Note: It depends on CPU frequency, generation, and so on.	20 Mbps	1	16	10 Mbps

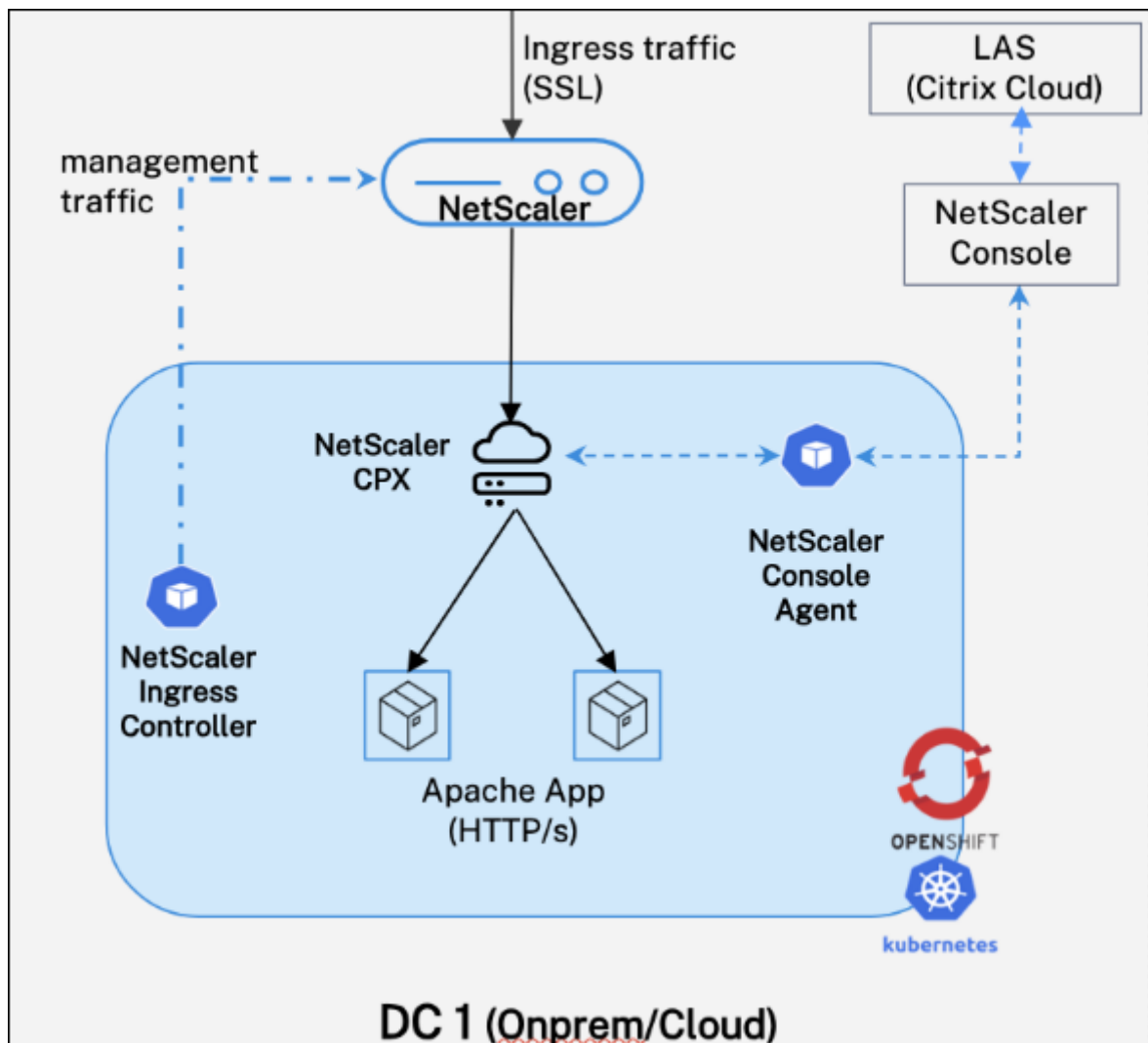
Note:

Citrix is currently working on a NetScaler CPX 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 NetScaler CPX licensing work?

NetScaler CPX pooled/CPL/UHMC licensing: A common license pool from which your NetScaler CPX 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 into the common pool, making the resources available to other instances that need them.

NetScaler CPX behavior: A single NetScaler CPX instance checking out up to 1 Gbps throughput, checks out only from the instance pool and not from the bandwidth license pool. NetScaler CPX operates in this way up to the 1 Gbps of bandwidth utilization. For example, if a NetScaler CPX instance consumes a 200 Mbps bandwidth, it uses the instance pool of license, instead of the bandwidth pool. However, if a NetScaler CPX instance consumes 1200 Mbps of throughput, the first 1000 Mbps is used from the instance pool and the remaining 200 Mbps is consumed from the bandwidth pool.



NetScaler CPX licensing workflow:

1. Set up NetScaler agent service or NetScaler agent on-prem with Citrix Cloud for LAS licensing.
2. Deploy the Console Container agent and check its status on NetScaler agent service or NetScaler agent on-prem.
3. Configure NetScaler CPX licensing and deploy NetScaler CPX.
 - a) On startup, NetScaler CPX sends a license registration request to the Console container agent, which relays it to NetScaler agent service or NetScaler agent on-prem.
 - b) NetScaler agent service or NetScaler agent on-prem registers NetScaler CPX and assigns bandwidth.
 - c) License is granted if pooled/UHMC/CPL license blobs are available.

How to license NetScaler CPX?

NetScaler CPX is licensed through the NetScaler agent service or on-prem. To license NetScaler CPX instances in a Kubernetes cluster, deploy a single Console container agent. One agent per cluster licenses all NetScaler CPX instances within that cluster.

Getting started with LASbased CPX licensing

1. Prerequisites: [NetScaler Console Service](#) or [NetScaler Console](#) on-premises with LAS based licensing must be enabled.
2. Deploy a container-based Console agent in the Kubernetes cluster where NetScaler CPX instances are deployed.
 - a) The following are the steps to deploy a container-based Console agent for NetScaler Console service customers:
 - i. Using [HELM chart](#)
 - ii. Using [OpenShift Operator](#)
 - b) The following are the steps to deploy a container-based Console agent for NetScaler Console on-prem customers.
 - i. Using [YAMLs](#).
 - ii. Using [OpenShift Operator](#).
3. Upgrade to NetScaler CPX 14.1–51.x or later to support LAS based licensing. Refer to the CPX HELM chart for the latest NetScaler CPX deployment. Update the following mandatory variables only from `values.yaml` of CPX HELM chart to support LAS-based licensing.
 - a) `licensingMode: las`
 - b) `ADMIP: <Console Container agent Service IP>` - Copy the container agent IP address from NetScaler Console on-prem or service from **Infrastructure > Instances Dashboard > Agents** section.

or

`ADMIP: <Console Container agent ServiceName.Namespace>` - Copy the container agent `HOSTNAME` from NetScaler Console on-prem or service from **Infrastructure > Instances Dashboard > Agents** section and append it with namespace in which container agent is deployed.
 - c) `loginSecret`: NetScaler Console Container agent login credentials details.

- i. In case of YAML/HELM based deployment, you have to create a Kubernetes secret based on the Application ID and password details provided while generating the console agent YAML from NetScaler Console on-prem or service.
- ii. In OpenShift Operator based deployment, you must have already created the NetScaler Console login secret (refer to `loginSecret` field). You can use the same secret name here.
- iii. `bandWidth`: The value in Mbps. For example, 500 Mbps.

```

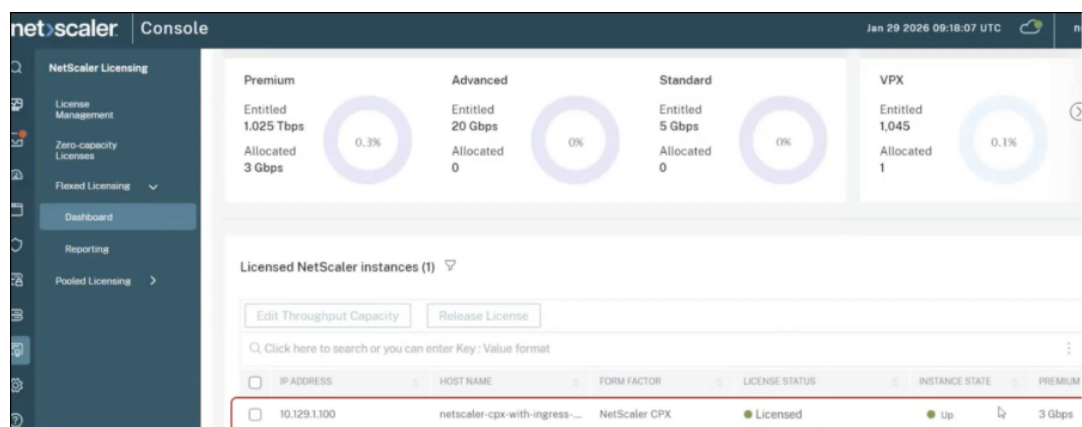
master netScaler-helm-charts / netScaler-cpx-with-ingress-controller / values.yaml

Blame 312 lines (277 loc) · 8.4 KB

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

```

- d) After successful NetScaler CPX licensing, NetScaler CPX instances start showing up in the NetScaler Console on-prem or service dashboard.



Troubleshooting NetScaler CPX licensing issues

- Verify NetScaler CPX licensing.

- Log in to NetScaler CPX using `oc exec -it <CPX POD name> --bash`, then run `cli_script.sh "sh capacity"` to view the actual bandwidth value.
- Run `cli_script.sh "sh license"` to check the `Licensing mode: LAS` value.
- Locating licensed NetScaler CPX instances in NetScaler Console service or on-prem.
 - On the NetScaler Console service or on-prem GUI, navigate to **NetScaler Licensing > Flexed licensing > Dashboard**, and search for the NetScaler Console form factor.

NetScaler CPX support for Kubernetes probes

September 7, 2026

Kubernetes manages container lifecycles through startup, liveness, and readiness probes to ensure that applications initialize correctly and remain healthy. NetScaler CPX now supports the readiness probe in addition to the existing startup and liveness probes. This support ensures that Kubernetes routes traffic to NetScaler CPX only when it is fully operational and, if configured, has successfully completed its license checkout.

By using these probes, you can improve application availability and ensure that failed or unresponsive instances are recreated promptly.

Some of the benefits are:

- Enhanced traffic management: Ensures NetScaler CPX receives traffic only when it is ready to handle requests.
- Operational reliability: Restarts unhealthy containers automatically to recover from failures.
- Licensing validation: Delays the ready state until license checkouts are successful when using the `LAS_CPX_BANDWIDTH` parameter.
- Precise initialization: Coordinates the transition from container startup to active health monitoring.

Note:

NetScaler CPX version 14.1-72.x or later is required to use the readiness probe. For detailed information on probe mechanics, see the official Kubernetes documentation.

Configure Kubernetes probes for NetScaler CPX

To configure probes for NetScaler CPX, define the probe parameters in your Kubernetes deployment YAML file.

To configure probes using the CLI

Edit your deployment YAML file to include the specific `exec` commands that Kubernetes uses to verify the state of the NetScaler CPX container.

Configure the startup probe: Add the following snippet to ensure that Kubernetes waits for the initialization process to finish. NetScaler CPX signals a successful start by creating the file. Configure the startup probe to monitor this path to ensure that initialization completes before liveness checks begin.

```
1
2   startupProbe:
3     exec:
4       command:
5         - ls
6         - /tmp/cpx_started
7     initialDelaySeconds: 30
8     periodSeconds: 5
9     failureThreshold: 20
10    successThreshold: 1
11    timeoutSeconds: 1
```

Configure the readiness probe: Add the following snippet to define when the pod is ready to accept traffic. CPX creates the `/tmp/cpx_ready` file when it is prepared to handle traffic. If you configure the `LAS_CPX_BANDWIDTH` parameter, CPX creates this file after a successful license checkout. Otherwise, it appears after the startup process finishes. Kubernetes uses this file for the readiness probe to determine when to route traffic to the pod.

```
1  readinessProbe:
2    exec:
3      command:
4        - /bin/ls
5        - /tmp/cpx_ready
6    failureThreshold: 15
7    initialDelaySeconds: 60
8    periodSeconds: 30
9    successThreshold: 1
10   timeoutSeconds: 1
```

Configure the liveness probe: Add the following snippet to continuously monitor the health of the container during runtime. Define a liveness probe in the deployment YAML to monitor CPX health during runtime. If the probe fails, Kubernetes restarts the container to recover from unhealthy states. This mechanism ensures application availability by automatically recreating unresponsive CPX instances.

```
1  livenessProbe:
2    exec:
3      command:
4        - /bin/ping
```

```
5      - -c 1
6      - 192.0.0.1
7      initialDelaySeconds: 10
8      periodSeconds: 5
9      failureThreshold: 3
10     successThreshold: 1
11     timeoutSeconds: 1
```

Deploying a NetScaler CPX Instance in Docker

September 7, 2026

NetScaler CPX instances are available as a Docker image file in the Quay container registry. To deploy an instance, download the NetScaler CPX 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 NetScaler CPX performance, you can define the number of processing engines that you want the NetScaler CPX 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: NetScaler CPX has issues while running on ubuntu version 16.04.5, kernel version 4.4.0-131-generic. So, it is not recommended to run NetScaler CPX on ubuntu version 16.04.5 kernel version 4.4.0-131-generic.

Note: The following kubelet and kube-proxy versions have some security vulnerabilities and it is not recommended to use NetScaler CPX with these versions:

- kubelet/kube-proxy v1.18.0-1.18.3
- kubelet/kube-proxy v1.17.0-1.17.6
- kubelet/kube-proxy <=1.16.10

For information on how to mitigate this vulnerability, see [Mitigate this vulnerability](#).

Downloading NetScaler CPX Image from Quay

You can download the NetScaler CPX image from the Quay container registry using the `docker pull` command and deploy it on your environment. Use the following command to download the NetScaler CPX image from the Quay container registry:

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

For example, if you want to download the version 13.0-64.35, then use the following command:

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

Use the following command to verify if the NetScaler CPX image is installed in docker images:

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

You can deploy the latest NetScaler CPX image from the [Quay container registry](#).

Deploying the NetScaler CPX Instance Using the docker run Command

On the host, you can install a NetScaler CPX instance in the Docker container by using the NetScaler CPX Docker image that you loaded onto the host. Using the `docker run` command, install the NetScaler CPX instance with the default NetScaler CPX configuration.

Install the NetScaler CPX instance on the Docker container by using the following **`docker run`** command:

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

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

This example creates a container named `mycpx` based on the NetScaler CPX Docker image.

The `-P` parameter is mandatory. It tells Docker to map the ports exposed in the container by the NetScaler CPX Docker image. That means mapping 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 NetScaler CPX 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 NetScaler CPX in Host mode of deployment then you need to provide all the system privileges to the NetScaler CPX.

To run the NetScaler CPX in bridge mode with a single or multiple cores, you can use the `--cap-add=NET_ADMIN` option instead of the privileged mode. The `--cap-add=NET_ADMIN` option enables you to run the NetScaler CPX container with full network privileges. With the `--cap-add=NET_ADMIN` option, use the `--sysctl kernel.shmmax=1073741824 --sysctl net.ipv6.conf.default.accept_dad=0 --sysctl kernel.core_pattern=/var/crash/core.%e.%p.%s` option in the docker run command to do the additional system control settings manually. These additional system control settings aren't done automatically under any non-privileged modes.

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

Ignore this option, if you are running NetScaler CPX in bridge or none network.

The `-e NS_NETMODE="HOST"` is a NetScaler CPX specific environment variable that allows you to specify that the NetScaler CPX is started in host mode. Once NetScaler CPX starts in host mode it configures 4 default iptables rules on a host machine for management access to the NetScaler CPX. 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

Ignore this environment variable, if you are running NetScaler CPX in bridge or none network.

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

Notes:

- 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.
- NetScaler CPX can support from 1 to 16 cores.

The `-e EULA = yes` is a mandatory NetScaler CPX specific environment variable, which is required to verify that you have read and understand the End User License Agreement (EULA) available at: [The Citrix licensing system components](#).

The `-e PLATFORM=CP1000` parameter specifies the NetScaler CPX license type.

If you are running Docker in a host network, you can assign dedicated network interfaces to the NetScaler CPX 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 NetScaler CPX container until you uninstall the NetScaler CPX container. When the NetScaler CPX container is provisioned all the assigned network interfaces are added to the NetScaler networking namespace.

Note

If you are running NetScaler CPX in 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 NetScaler CPX container to use the updated network.

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

The `-e CPX_CONFIG` is a NetScaler CPX specific environment variable that enables you to control the throughput performance of the NetScaler CPX container. When the NetScaler CPX 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 NetScaler CPX container in such scenarios. You need to provide following values to the `CPX_CONFIG` environment variable in JSON format:

- If you want the NetScaler CPX container to yield CPU in idle scenarios, define { "YIELD" : "Yes" }
- If you want the NetScaler CPX container to avoid yielding the CPU in idle scenarios so that you can get high throughput performance, define { "YIELD" : "No" }

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

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

The `-v` parameter is an optional parameter that specifies the mount point of the NetScaler CPX 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 NetScaler CPX 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.

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

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 a Lighter Version of NetScaler CPX Using the `docker run` Command

NetScaler provides a lighter version of NetScaler CPX which consumes lesser runtime memory. The lighter version of NetScaler CPX can be deployed as a sidecar in service-mesh deployments.

The lighter version of NetScaler CPX 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®

To instantiate the lighter version of NetScaler CPX, set the `NS_CPX_LITE` environment variable while executing the `Docker run` command.

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

The following example creates a lightweight container based on the NetScaler CPX image.

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

By default, logging using `newnslog` is disabled on the lighter version of NetScaler CPX. To enable it, you must set the `NS_ENABLE_NEWNSLOG` environment variable to 1 while bringing up the lighter version of NetScaler CPX.

The following example shows how to enable logging using `newnslog` while deploying the lighter version of NetScaler CPX.

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

Note: The lighter version of CPX supports only single core (CPX_CORES=1).

Deploying NetScaler CPX Instances by Using Docker Compose

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

Important

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

To provision multiple NetScaler CPX 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 NetScaler CPX image.
 - **privileged: true** provides all root privileges to the NetScaler CPX instance.
 - **cap_add** provides network privileges to the NetScaler CPX instance.
 - **<host_directory_path>** denotes the directory on the docker host that you want to mount for the NetScaler CPX instance.
 - **<number_processing_engine>** is the number of processing engines that you want the NetScaler CPX 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:

```
1 <container_name>:
2   image: <repository>:<tag>
3   ports:
4       - 9443
5       - 22
6       - 9080
7       - 161/udp
8   tty: true
9   cap_add:
10      - NET_ADMIN
11   ulimits:
12      core: -1
13   volumes:
14      - <host_directory_path>:/cpx
```

```
15     environment:
16         - EULA=yes
17         - CPX_CORES=<number_processing_engine>
18         - CPX_CONFIG='{
19     "YIELD":"Yes" }
20     '
```

```
1 CPX_0:
2     image: quay.io/citrix/citrix-k8s-cpx-ingress:13.1-37.38
3     ports:
4         - 9443
5         - 22
6         - 9080
7         - 161/udp
8     tty: true
9     cap_add:
10         - NET_ADMIN
11     ulimits:
12         core: -1
13     volumes:
14         - /root/test:/cpx
15     environment:
16         - CPX_CORES=2
17         - EULA=yes
```

Adding NetScaler CPX Instances to NetScaler Console

September 7, 2026

You must add the NetScaler CPX instances installed on a Docker host to NetScaler 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 later.

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 instances that you want to add to NetScaler Console. For each instance type, a default profile is available. For example, the [ns-root-profile](#) is the default profile for NetScaler 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 profile, and then rediscover the instance.

Prerequisites

Make sure that you have:

- Installed the NetScaler Console software on Citrix XenServer. For more information, see [NetScaler Console Documentation](#).
- Installed the NetScaler CPX instances on a Docker host.

To add NetScaler CPX instances to ADM:

1. In a web browser, type the IP address of the **NetScaler 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 > NetScaler** and click **CPX** tab.
4. Click **Add** to add new CPX instances in NetScaler Console.
5. The **Add NetScaler CPX** page opens. Enter the values for the following parameters:
 - a) 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.
 - b) Select the profile of the CPX instance.
 - c) Select the site where the instances are to be deployed.
 - d) Select the agent.
 - e) As an option, you can enter the key-value pair to the instance. Adding a key-value pair makes it easy for you to search for the instance later.

6. Click **OK**.

Note

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

Adding NetScaler CPX instances to NetScaler Console using environment variables

You can also add the NetScaler CPX instances to NetScaler Console using environment variables. To add instances, you must configure the following environment variables for the NetScaler CPX 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** - (NetScaler CPX pod IP address is not routable from outside.)
- **NS_MGMT_USER** -ADM username
- **NS_MGMT_PASS** -ADM password

The following is an example `docker run` command for adding a NetScaler CPX instance to NetScaler Console.

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

Adding NetScaler CPX instances to NetScaler Console using Kubernetes ConfigMaps

NetScaler CPX supports registration with NetScaler Console by using volume mounted files through Kubernetes ConfigMaps. To enable this way of registration, NetScaler CPX requires some environment variables which are to be specified along with some volume mounted files through ConfigMaps and Secrets.

The following are the required environment variables and their description:

- `NS_HTTP_PORT` - Specifies mapped HTTP port on node.
- `NS_HTTPS_PORT` - Specifies mapped HTTPS port on node.
- `NS_SSH_PORT` - Specifies mapped SSH port on node.
- `NS_SNMP_PORT` - Specifies mapped SNMP port on node.

Apart from the listed environment variables, NetScaler CPX requires information about the ADM agent with which it has to register. This information contains ADM agent's IP address or FQDN details and credentials. NetScaler CPX acquires this information from the volume mounted files. A ConfigMap containing the IP address or FQDN is mounted as a file in the file-system of the NetScaler CPX instance. A Kubernetes secret containing credentials for the ADM agent is also mounted as a file in the NetScaler CPX instance's file-system. With all the information required for registration, NetScaler CPX attempts to register with the ADM agent.

The following is an example of a NetScaler CPX YAML file snippet with the ConfigMap and Secret mounted as files:

```
1  ...
2  env:
3  - name: "EULA"
4    value: "yes"
5  - name: "NS_HTTP_PORT"
6    value: "9080"
7  - name: "NS_HTTPS_PORT"
8    value: "9443"
9  - name: "NS_SSH_PORT"
10   value: "22"
11 - name: "NS_SNMP_PORT"
12   value: "161"
13 - name: "KUBERNETES_TASK_ID"
```

```

14     value: ""
15     ...
16     volumeMounts:
17     ...
18     - mountPath: /var/admininfo/server/
19       name: adm-agent-config
20     - mountPath: /var/admininfo/credentials/
21       name: adm-agent-user
22     ...
23     volumes:
24     ...
25     - name: adm-agent-config
26       configMap:
27         name: adm-agent-config
28     - name: adm-agent-user
29       secret:
30         secretName: adm-secret

```

In the preceding example, a ConfigMap named `adm-agent-config` and a secret `adm-agent-user` are consumed. The following is an example for creating the required ConfigMap and Secret.

ConfigMap: The ConfigMap is created from a file named `adm_reg_envs`. The file requires the IP address or FQDN of the ADM agent in the following format:

```
1 NS_MGMT_SERVER=adm-agent
```

In the preceding format, the `adm-agent` is the FQDN of the ADM agent to which the NetScaler CPX instance needs to be registered.

Use the following command to create a ConfigMap:

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

Note: The file name must have the `adm_reg_envs` variable and it must be mounted to the path: `/var/admininfo/server/`.

Secret: Use the following command to create a Kubernetes secret. In the following command, `user123` is the user name of the ADM agent and `pass123` is the password.

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

A NetScaler CPX instance can be deployed in a Kubernetes cluster with the required environment variables and volume mounted files even before deploying the ADM agent in the cluster. If you deploy a NetScaler CPX instance before deploying the ADM agent, NetScaler CPX keeps on trying to get registered until the ADM agent is deployed. Once the ADM agent is deployed, the NetScaler CPX instance uses the configuration data provided through the environment variables and volume mounted files to register with the ADM agent. It helps you to avoid the redeployment of NetScaler CPX with the configuration information.

A NetScaler CPX instance, that is already registered with an ADM agent, can dynamically change the registration to another ADM agent after a change in the configuration. For this, you can update configuration information in the ConfigMap and the Secret for the already deployed NetScaler CPX. You must update the file from which the ConfigMap is created with the IP address or FQDN of the new ADM agent and delete the old ConfigMap and, then create a new ConfigMap. Similarly, the existing secret must be deleted and a new secret must be created with the credentials for the new ADM agent.

NetScaler CPX License Aggregator

September 7, 2026

Currently, NetScaler CPXs obtain licenses from the NetScaler Console server. In a Kubernetes environment, NetScaler CPXs can dynamically come up or go down. If a NetScaler CPX unexpectedly goes down, it takes few minutes for the NetScaler Console server to reclaim the license. The NetScaler Console server must be able to reclaim such licenses immediately as and when NetScaler CPXs go down so that the same license can be allocated to another upcoming NetScaler CPX. Also, if the NetScaler Console server is not reachable for any reason, you cannot license new NetScaler CPXs in the cluster.

NetScaler CPX License Aggregator is a Kubernetes service provided by NetScaler. This service acts as a local provider for NetScaler CPX licensing within a Kubernetes cluster. The NetScaler CPX License Aggregator service deployed in a Kubernetes cluster can act as an intermediary between NetScaler CPXs and the ADM licensing server and keep track of NetScaler CPXs and the allotted licenses. With the NetScaler CPX License Aggregator service, the NetScaler Console server can reclaim licenses immediately as and when NetScaler CPXs goes down.

In a Kubernetes cluster, the NetScaler CPX License Aggregator service supports both NetScaler CPX as sidecar and standalone deployments.

Note:

Licensing using NetScaler CPX License Aggregator requires NetScaler CPX 13.1-30.x or later. NetScaler CPX License Aggregator does not support licensing the older versions of NetScaler CPX.

Key benefits of NetScaler CPX License Aggregator

The following are the key benefits of using NetScaler CPX License Aggregator:

- Scalability:
A NetScaler Console licensing server can support only up to 10000 NetScaler CPX deployments.

With the introduction of NetScaler CPX License Aggregator service, each Kubernetes cluster can act as a single client to the NetScaler Console licensing server. Hence, you can scale many NetScaler CPXs with a single NetScaler Console licensing server.

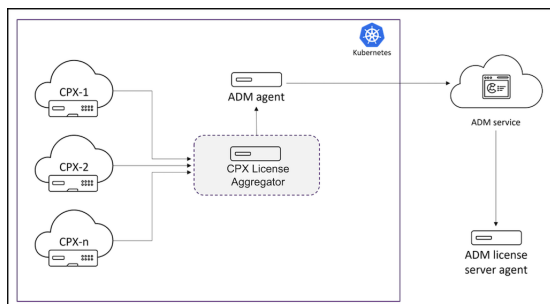
- **Resource optimization:**

The NetScaler CPX License Aggregator service also supports cluster-wide licensing capability and it can also check out licenses from the NetScaler Console server on a need basis. NetScaler CPX License Aggregator can return licenses back to the NetScaler Console server.

NetScaler CPX License Aggregator can handle ungraceful termination of NetScaler CPXs and reclaim licenses from such NetScaler CPXs after the configured waiting period.

Deployment topology

The following diagram shows a NetScaler CPX License Aggregator deployment within a Kubernetes cluster.



In this diagram:

- **CPX** means NetScaler CPX
- **CPX License Aggregator** means NetScaler CPX License Aggregator

In this sample deployment, a NetScaler CPX License Aggregator service is deployed within the Kubernetes cluster along with NetScaler CPXs and NetScaler agent. The NetScaler CPX License Aggregator service acts as an intermediary between NetScaler CPXs and NetScaler agent and monitors all NetScaler CPXs within the cluster and manages licenses to them.

Deploy NetScaler CPX License Aggregator using Helm charts

Prerequisites

The following prerequisites apply:

- You need Kubernetes version 1.16 and later.
- You need the Helm version 3.x or later.

- You must get the IP address of the license server that has the license for NetScaler CPX.
- You must provide a password for the Redis database in NetScaler CPX License Aggregator. You can provide the database password using the Kubernetes secret and the following command can be used to create the secret:

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

Deployment using Helm charts

Do the following steps to deploy NetScaler CPX License Aggregator using Helm charts depending on the type of NetScaler CPX license. For more information about the different type of NetScaler CPX licenses, see [NetScaler CPX licensing](#).

Installing the Helm chart Add the NetScaler CPX License Aggregator Helm chart repository using the following command:

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

Installing NetScaler CPX License Aggregator to manage bandwidth pooled licenses Use one of the following commands depending on the type of NetScaler CPX pooled license that you have. In these commands, `my-release` is used as the release name.

Note:

By default, the Helm chart installs the recommended RBAC roles and role bindings.

For platinum bandwidth license:

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

For enterprise bandwidth edition:

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

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

For standard bandwidth edition:

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

These commands deploy the NetScaler CPX License Aggregator on the Kubernetes cluster with the default configuration. You can configure the parameters at the time of installation. For more information, see the **NetScaler CPX License Aggregator configuration** section in the [Helm chart GitHub repository](#) that lists the mandatory and optional parameters that you can configure during installation.

Installing NetScaler CPX License Aggregator to manage vCPU Licenses

Use one of the following commands depending on the type of NetScaler CPX vCPU license that you have. In these commands, `my-release` is used as the release name.

Note:

By default, the Helm chart installs the recommended RBAC roles and role bindings.

For platinum vCPU edition:

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

For enterprise vCPU edition:

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

For standard vCPU edition:

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

Installing NetScaler CPX License Aggregator to manage multiple licenses

If you need NetScaler CPX license aggregator to manage multiple types of licenses, then the relevant arguments of those licenses must be specified in the Helm command.

For example:

To deploy NetScaler CPX License Aggregator for `pooled platinum bandwidth edition` and `vCPU platinum edition` licenses:

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

Configuring NetScaler CPX to get license from NetScaler CPX License Aggregator

When you use NetScaler CPX License Aggregator for licensing NetScaler CPX, the environment variable `CLA` need to be provided in the NetScaler CPX deployment YAML.

The `ipaddress` or `domainname` using which NetScaler CPX License Aggregator can be accessed must be provided in this environment variable as follows:

```
1 env:
2   - name: "CLA"
3     value: "192.0.2.2"
```

Or

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

You must provide the following environment variables as well in the NetScaler CPX YAML.

- **POD_NAME**: Specifies the name of the pod. The name of the pod is exposed to NetScaler CPX as an environment variable.

- **POD_NAMESPACE**: Specifies the namespace of the pod. The namespace of the pod is exposed to NetScaler CPX as an environment variable.
- **Bandwidth**: Specifies the bandwidth in Mbps for allocating to NetScaler CPX.
- **Edition**: Specifies the license edition. The supported values include Standard, Platinum, and Enterprise.
- **CPX_CORES**: Specifies the number of cores that you want to run for NetScaler CPX.

For more information on different NetScaler CPX licensing options, see [NetScaler CPX licensing](#).

The following shows a sample configuration with these environment variables:

```
1      - name: POD_NAME
2        valueFrom:
3          fieldRef:
4            apiVersion: v1
5            fieldPath: metadata.name
6      - name: POD_NAMESPACE
7        valueFrom:
8          fieldRef:
9            apiVersion: v1
10         fieldPath: metadata.namespace
11     - name: "BANDWIDTH"
12       value: 1000
13     - name: "CPX_CORES"
14       value: 1
15     - name: "EDITION"
16       value: PLATINUM
```

You also need to add the following label to the NetScaler CPX YAML:

```
1      labels:
2        adc: citrix®
```

For a sample deployment of NetScaler CPX License Aggregator, see [NetScaler CPX License Aggregator: Sample deployment](#).

Configuring NetScaler CPX

September 7, 2026

You can configure a NetScaler CPX instance by accessing the CLI prompt through the Linux Docker host, or by using the NetScaler NITRO APIs.

Configuring a NetScaler CPX 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 NetScaler CPX 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**.

Support for using a non-default password in NetScaler CPX

NetScaler CPX supports using a non-default password for the root account, that is **nsroot**. A default password is generated and assigned to the user once NetScaler CPX has been deployed. This default password is also updated for SSH users: **root** and **nsroot**. You can change this default password manually. You can also reset the default SSH password for **root** and **nsroot** user accounts manually. Citrix recommends changing this password manually to preserve the security of the system.

Once you reset your password, the new password is used for NITRO API communications and **cli_script.sh** executions.

The default root account password is stored in plain text in the `/var/deviceinfo/random_id` file in the NetScaler CPX file system.

Use the following syntax for running **cli_script.sh** with the credentials:

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

For example, to run **cli_script.sh** for displaying IP addresses with user **nsroot** and password **Citrix123**, use the following:

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

Configuring a NetScaler CPX Instance by Using the NITRO API

You can use the NetScaler NITRO API to configure NetScaler CPX instances.

To configure NetScaler CPX 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 NetScaler CPX, use `CPX IP address:port` where `netScaler-ip-address` is mentioned.

Configuring a NetScaler CPX Instance by Using Jobs

You can configure NetScaler CPX instances by creating and running jobs in NetScaler Console. 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. NetScaler Console then displays the corresponding CLI commands for you to use on your NetScaler CPX instance. After you select the configuration, you must then select **NetScaler CPX instances** on which you want to load the configuration, specify the variable values, and run the job.

To configure NetScaler CPX instances by using Jobs:

1. Log on to NetScaler Console 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*

Instance Type*

Configuration Editor

Configuration Source

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

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

- Select the NetScaler CPX 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

- Specify the execution settings and click Finish to run the commands on the NetScaler CPX instance. If you want to save the configuration and run it later, click **Save and Exit**.

The screenshot displays the 'Execute' step of a configuration process in the NetScaler CPX interface. At the top, there are four tabs: 'Select Configuration' (gear icon), 'Select Instances' (stack of papers icon), 'Specify Variable Values' (play button icon), and 'Execute' (code icon). Below the tabs, a message states: 'You can either run the commands now or schedule to run the commands at a later time.' The main configuration area includes a dropdown for 'On Command Failure*' set to 'Ignore error and continue', a dropdown for 'Execution Mode*' set to 'Now', and a section titled 'Execution Settings' with two radio buttons: 'Execute in Sequence' (selected) and 'Execute in Parallel'. Below this is a section 'Receive Execution Report Through' with a checkbox for 'Email'. At the bottom, there are four buttons: 'Cancel', 'Back' (with a left arrow), 'Finish' (highlighted in blue), and 'Save and Exit'.

Configuring AppFlow on a NetScaler CPX instance

September 8, 2026

You can configure AppFlow feature on a NetScaler CPX 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 NetScaler Console where you can view real-time and historical reports for all your applications.

To configure AppFlow, first, you must enable the AppFlow feature. Then, you specify the collectors to which the flow records are sent. After that, you define actions, which are sets of configured collectors. Then you configure one or more policies and associate an action to each policy. The policy tells the NetScaler CPX to select requests the flow records of which are sent to the associated action. Finally, you bind each policy either globally or to the specific virtual server to put it into effect.

You can further set AppFlow parameters to specify the template refresh interval and to enable the exporting of [httpURL](#), [httpCookie](#), and [httpReferer](#) information. On each collector, you must specify the NetScaler CPX IP address as the address of the exporter.

The configuration utility provides tools that help users define the policies and actions. It determines exactly how the NetScaler CPX export records for a particular flow to a set of collectors(action.) The command line interface provides a corresponding set of CLI-based commands for experienced users who prefer a command line.

Before you can monitor the records, you must add the NetScaler CPX instance to the NetScaler Console. For more information about adding a NetScaler CPX instance to NetScaler Console, see [Installing a NetScaler CPX Instance by Using NetScaler Console](#).

Enable AppFlow®

To use the AppFlow feature, you must first enable it.

To enable the AppFlow feature by using the command line interface:

Run the following commands:

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

Specify a Collector

A collector receives AppFlow records generated by the NetScaler. To send the AppFlow records, you must specify at least one collector. By default, the collector listens to IPFIX messages on UDP port 4739. You can change the default port, when configuring the collector.

To specify a collector by using the command line interface:

Use the following commands to add a collector:

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

To verify the configuration, use the following command:

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

To specify multiple collectors by using the command line interface:

Use the following commands to add and send the same data to multiple collectors:

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

Configuring an AppFlow Action

An AppFlow action is a set collector, to which the flow records are sent if the associated AppFlow policy matches.

Use the following commands to configure an AppFlow action:

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

To verify the configuration, use the following command:

```
1 show appflow action
```

Configuring an AppFlow Policy

After you configure an AppFlow action, you must next configure an AppFlow policy. An AppFlow policy is based on a rule, which consists of one or more expressions.

To configure an AppFlow policy by using the command line interface:

At the command prompt, type the following command to add an AppFlow policy and verify the configuration:

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

Binding an AppFlow Policy

To put a policy into effect, you must bind it either globally, so that it applies to all traffic that flows through the NetScaler CPX.

To globally bind an AppFlow policy by using the command line interface:

Use the following command to globally bind an AppFlow policy:

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

Verify the configuration using the following command:

```
1 show appflow global
```

Configuring NetScaler CPX Using a Configuration File

September 7, 2026

Instead of using command line interface (`cli_script.sh`), NITRO API, or NetScaler Console configuration jobs to configure the NetScaler CPX, you can configure the NetScaler CPX using a static configuration file while deploying the NetScaler CPX instance.

You can provide a static configuration file as an input file while deploying the NetScaler CPX container. During NetScaler CPX 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 NetScaler CPX container.

Structure of the static configuration file

As mentioned earlier, when NetScaler CPX 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 NetScaler-specific configuration on NetScaler CPX. Under the `#Shell Commands` tag, you must add the shell commands that you want to run on NetScaler CPX.

During the NetScaler CPX 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 NetScaler CPX container with the configuration file, the failures are logged in the `ns.log` file in the container.
- When you reboot the NetScaler CPX container the configuration file is reapplied on the container.

```
1 #NetScaler Commands
2
3 add lb vserver v1 http 1.1.1.1 80
4
```

```
5 add service s1 2.2.2.2 http 80
6
7 bind lb vserver v1 s1
8
9 #Shell Commands
10
11 touch /etc/a.txt
12
13 echo "this is a" > /etc/a.txt
14
15 #NetScaler Commands
16
17 add lb vserver v2 http
18
19 #Shell Commands
20
21 echo "this is a 1" >> /etc/a.txt
22
23 #NetScaler Commands
24
25 add lb vserver v3 http
26
27 #This is a test configuration file
```

To install a NetScaler CPX container and to dynamically configure the NetScaler CPX container based on a configuration file, mount the static configuration file using the `-v` option in the `docker run` command:

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

Dynamic Routing support in NetScaler CPX

September 7, 2026

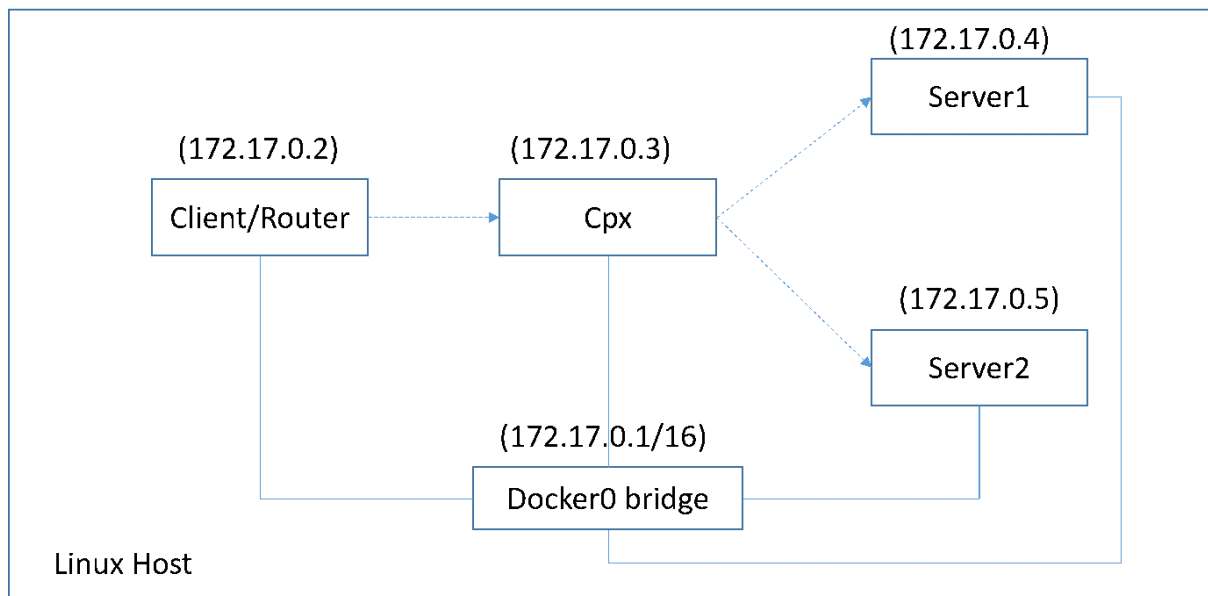
NetScaler CPX supports the BGP dynamic routing protocol. The key objective of the dynamic routing protocol is to advertise the virtual server's IP address based on the health of the services, bound to the virtual server. It helps an upstream router to choose the best among multiple routes to a topographically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

For information about the non-default password in NetScaler CPX, see the [Support for using a non-default password in NetScaler CPX](#) section in the [Configuring NetScaler CPX](#) document.

In a single host network, the client, the servers, and the NetScaler CPX instance are deployed as containers on the same Docker host. All the containers are connected through the `docker0` bridge. In this

environment, the NetScaler CPX instance acts as a proxy for the applications provisioned as containers on the same Docker host. For information about NetScaler CPX 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 NetScaler CPX in single Docker host with the bridge-networking mode.

Configure BGP based Route Health Injection using REST API on NetScaler CPX

1. Create a container from the NetScaler CPX image using the following command:

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

For example:

```
1 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:

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

3. Enable the BGP feature using the following command:

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

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

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

5. Add the virtual server using the following command:

```
1 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:

```
1 cli_script.sh "set ns ip <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:

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

Specify the command as the following:

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

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

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

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

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

Sample output:

```
1 root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/  
nitro/v1/config/routerDynamicRouting/bgpRouter  
2 {  
3   "errorcode": 0, "message": "Done", "severity": "NONE", "  
   routerDynamicRouting":{  
4   "bgpRouter":[{  
5     "localAS": 100, "routerId": "172.17.0.3", "afParams": [ {  
6     "addressFamily": "ipv4" }  
7   , {  
8     "addressFamily": "ipv6" }  
9   ], "neighbor": [ {  
10    "address": "172.17.0.2", "remoteAS": 101, "ASOriginationInterval  
      ": 15, "advertisementInterval": 30, "holdTimer": 90, "  
      keepaliveTimer": 30, "state": "Connect", "singlehopBfd":  
        false, "multihopBfd": false, "afParams": [ {  
11    "addressFamily": "ipv4", "activate": true }  
12    , {  
13    "addressFamily": "ipv6", "activate": false }  
14  ]
```

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

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

12. Save the configuration using the following command:

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

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

Configuring Docker Logging Drivers

September 7, 2026

Docker includes logging mechanisms called “logging drivers” to help you get information from the running containers. You can configure a NetScaler CPX container to forward logs that it generates to the docker logging drivers. For more information on docker logging drivers, see [Configure logging drivers](#).

By default, all logs generated by the NetScaler CPX container are stored in `/cpx/log/ns.log` file on the docker host. When you start the NetScaler CPX 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 NetScaler CPX container is configured to forward all the generated logs using syslog as logging driver.

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

Similarly, in the following example the NetScaler CPX container is configured to forward all the generated logs using Splunk as logging driver.

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

Upgrading a NetScaler CPX Instance

September 7, 2026

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:

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

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 and 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:

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

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.

```
1 root@ubuntu:~# docker stop mycpx
2 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.

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

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

```
1  ...
2  root@ubuntu:~# docker ps
3
4  CONTAINER ID        IMAGE               COMMAND             PORTS
5  CREATED            STATUS             NAMES
6  ead12ec4e965       cpx:13.0-x.x      "/bin/sh -c 'bash -C " 5
7  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.

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

Using Wildcard Virtual Servers in NetScaler CPX Instance

September 7, 2026

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 the 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, which is replaced 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 the primary NetScaler CPX instance. The configuration sync between nodes configures the wildcard virtual server on the secondary NetScaler CPX instance. It eliminates the need for configuring the virtual server on the NSIP assigned by the Docker engine to the NetScaler CPX instances.

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:

```
1      add lb vserver <name> <serviceType> 127.0.0.1 <port>
2
3      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:

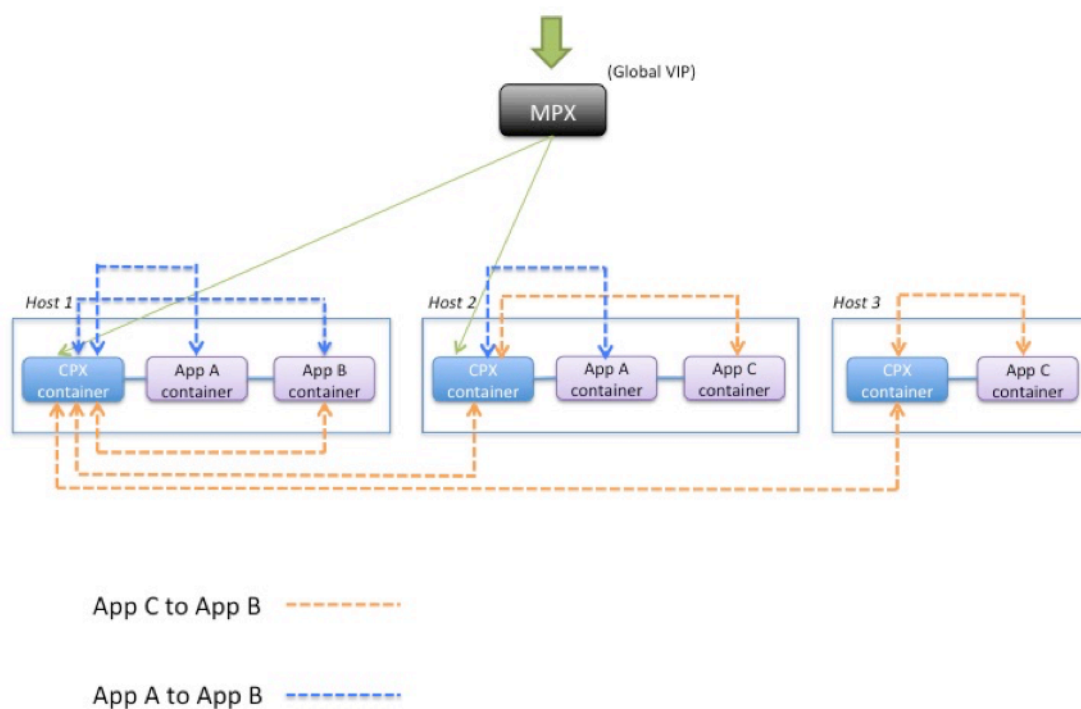
```
1      add cs vserver <name> <serviceType> 127.0.0.1 <port>
2
3      add cs vserver testcsvserver HTTP 127.0.0.1 30000
```

Deploy NetScaler CPX as a Proxy to Enable East-West Traffic Flow

September 8, 2026

In this deployment, the NetScaler CPX instance acts as a proxy to enable communication between application containers residing on multiple hosts. The NetScaler CPX instance is provisioned along with the applications in multiple hosts and provides the shortest path for communication.

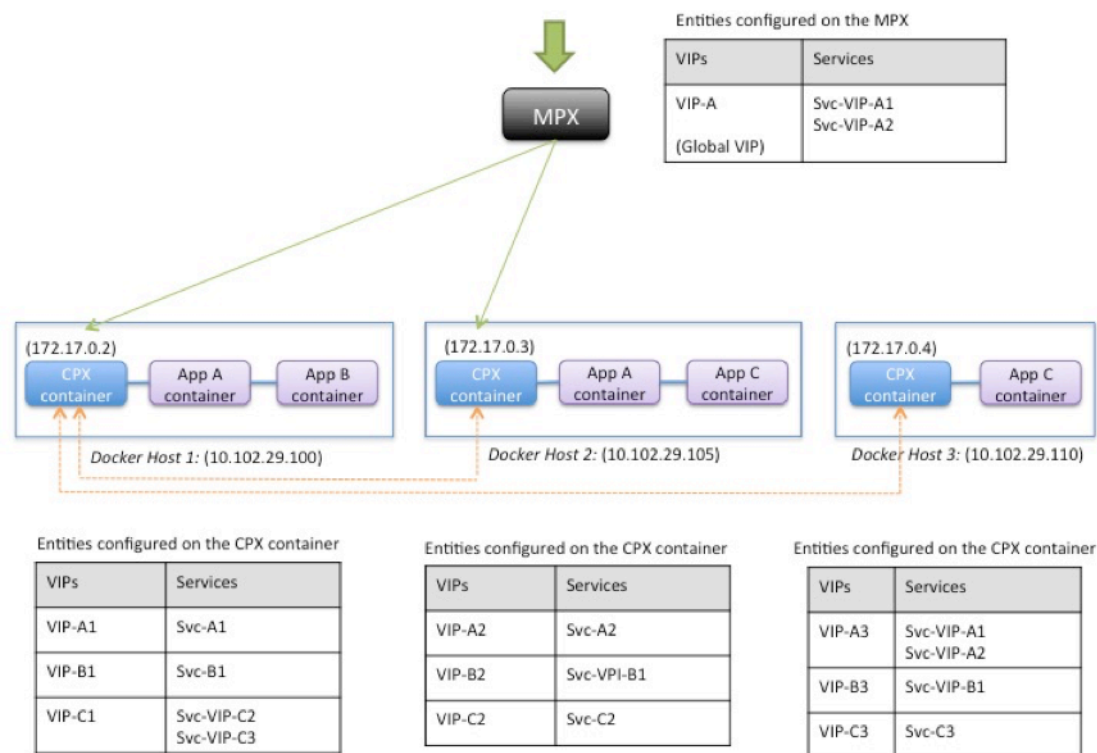
The following image illustrates traffic flow between two applications through the NetScaler CPX instances.



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 NetScaler CPX container on the same host as app C. Then, the NetScaler CPX container passes the traffic to the NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX container on all three Docker hosts:

```
1 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 Console or by using NITRO APIs.

On NetScaler CPX instance on Docker Host 1:

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

On the NetScaler CPX instance on Docker host 2:

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

On the NetScaler CPX instance on Docker host 3:

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

Deploy NetScaler CPX in a Single Host Network

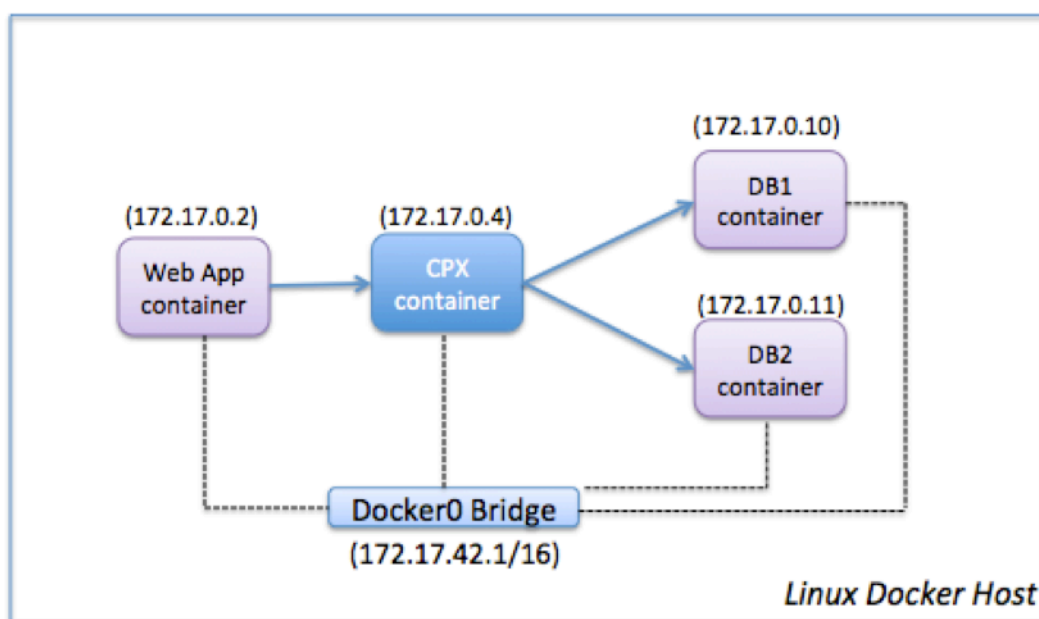
September 8, 2026

In a single host network, the NetScaler CPX instance acts as a proxy between application containers on the same host. In this capacity, the NetScaler CPX 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 NetScaler CPX instance are deployed as containers on the same Docker host. All the containers are connected through docker0 bridge.

In this environment, the NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX, you have to first configure two services on the NetScaler CPX container to represent the two servers. Then, configure a virtual server by using the NetScaler CPX IP address and a non-standard HTTP port (such as 81) because the NetScaler CPX 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 Console or by using NITRO APIs:

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

Deploy NetScaler CPX in a Multi-Host Network

September 8, 2026

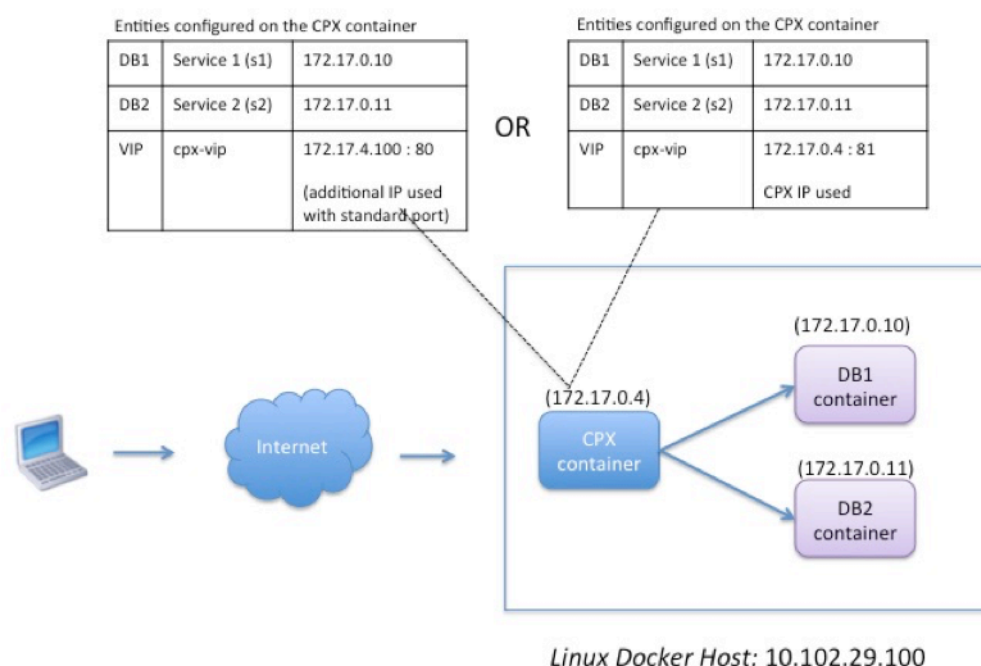
A NetScaler CPX instance in a multi-host network may be configured in a production deployment in the datacenter where it provides load balancing functions. It can further provide monitoring functions and analytics data.

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

Topology 1: NetScaler CPX and Backend Servers on Same Host; Client on a Different Network

In this topology, the NetScaler CPX 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 NetScaler CPX instance load balances a set of container-based applications or servers.

The following diagram illustrates this topology.



In this example, the NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX 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 Console or by using NITRO APIs:

```
1 add service s1 172.17.0.10 HTTP 80
```

```
2    add service s2 172.17.0.11 HTTP 80
3    add lb vserver cpx-vip HTTP 172.17.4.100 80
4    bind lb vserver cpx-vip s1
5    bind lb vserver cpx-vip s2
```

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

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

Method 2: Using the NetScaler CPX IP address for the VIP and by configuring port mapping:

1. Configure the VIP and the two services on the NetScaler CPX 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 NetScaler CPX container on all three Docker hosts:

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

After the NetScaler CPX instance is provisioned, run the following commands either by using the Jobs feature in NetScaler Console or by using NITRO APIs:

```
1    add service s1 172.17.0.10 http 80
2    add service s2 172.17.0.11 http 80
3    add lb vserver cpx-vip HTTP 172.17.0.4 81
4    bind lb vserver cpx-vip s1
5    bind lb vserver cpx-vip s2
```

Note:

If you have not configured port mapping during provisioning of the NetScaler CPX instance, then configure a NAT rule by running the following commands at the Linux shell prompt:

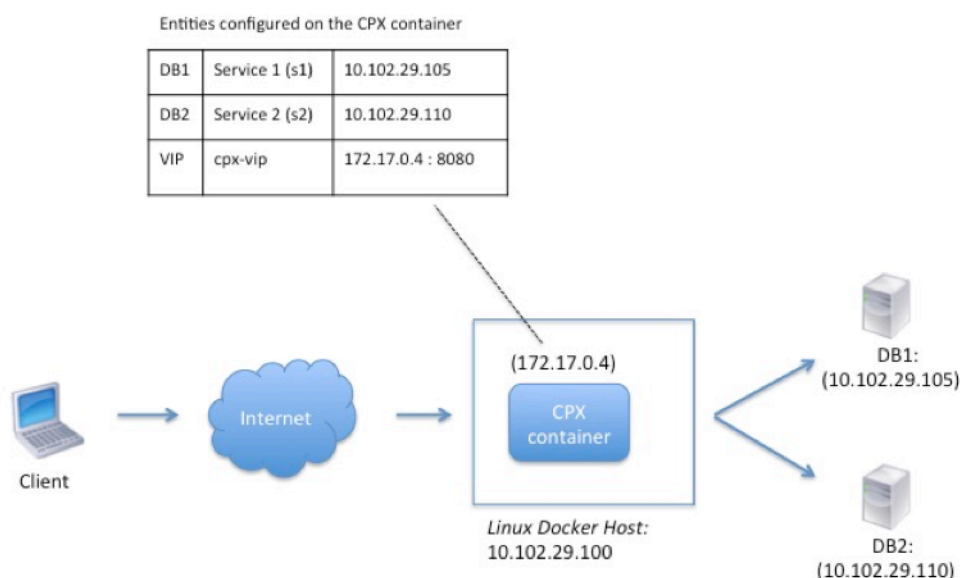
```
iptables -t nat -A PREROUTING -p tcp -m addrtype --dst-type LOCAL -m tcp --dport 50000 -j
DNAT --to-destination 172.17.0.4:81
```

Topology 2: NetScaler CPX with Physical Servers and Client

In this topology, only the NetScaler CPX 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 NetScaler CPX instance to load balance traffic across the physical servers.

The following figure illustrates this topology.



In this example, the NetScaler CPX 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 NetScaler CPX, which distributes it across the two servers.

To enable this communication between the client and the servers through NetScaler CPX, you have to first configure port mapping while creating the NetScaler CPX container. Then, configure the two services on the NetScaler CPX container to represent the two servers. And finally, configure a virtual server by using the NetScaler CPX 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 NetScaler CPX container:

```
1 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 Console or by using NITRO APIs:

```

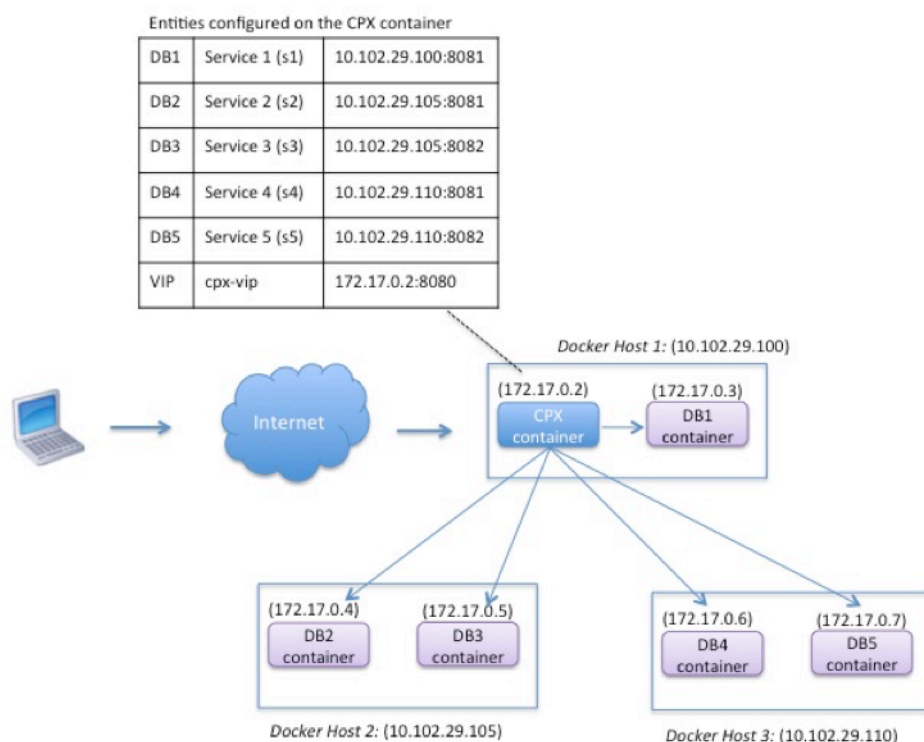
1  add service s1 HTTP 10.102.29.105 80
2  add service s2 HTTP 10.102.29.110 80
3  add lb vserver cpx-vip HTTP 172.17.0.4 8080
4  bind lb vserver cpx-vip s1
5  bind lb vserver cpx-vip s2

```

Topology 3: NetScaler CPX and Servers Provisioned on Different Hosts

In this topology, the NetScaler CPX 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 NetScaler CPX instance load balances a set of container-based applications or servers.

The following diagram illustrates this topology.



In this example, the NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX instance to receive the client request. This VIP should be represented by the NetScaler CPX 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 NetScaler CPX container:

```
1    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 Console or by using NITRO APIs:

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

Deploy NetScaler CPX with direct access to the network

September 7, 2026

In bridge networking mode, you can configure NetScaler CPX instance to have direct access to the network. In this scenario, the incoming traffic is directly received on the NetScaler CPX 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 NetScaler CPX 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:

```
1 ip addr add 10.102.29.100/24 dev docker0;
2 ip addr del 10.102.29.100/24 dev eth0;
3 brctl addif docker0 eth0;
4 ip route del default;
5 ip route add default via 10.102.29.1 dev docker0
```

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

```
1 add service s1 172.17.0.8 http 80
2 add service s2 172.17.0.9 http 80
3 add lb vserver cpx-vip HTTP 10.102.29.102 80
4 bind lb vserver cpx-vip s1
5 bind lb vserver cpx-vip s2
```

Configure NetScaler CPX in Kubernetes Using ConfigMaps

September 7, 2026

In Kubernetes, you can configure the NetScaler CPX instance using ConfigMaps. Using ConfigMaps you can dynamically configure the NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX instance. Under the `#Shell Commands` tag, you must add the shell commands that you want to run on the NetScaler CPX 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 NetScaler CPX container with the configuration file, the failures are logged in the `ns.log` file.
- After the NetScaler CPX instance starts, if you change the ConfigMap, the updated configu-

ration is applied only when the NetScaler CPX instance is restarted.

The following is a sample configuration file:

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

```
1 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 NetScaler CPX instance.

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

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

Sample:

```
1  apiVersion: v1
2  data:
3    cpx.conf: |
4      #NetScaler Commands
5        add lb vserver v1 http 1.1.1.1 80
6        add service s1 2.2.2.2 http 80
7        bind lb vserver v1 s1
8      #Shell Commands
9        touch /etc/a.txt
10       echo "this is a" > /etc/a.txt
11       echo "this is the file" >> /etc/a.txt
12       ls >> /etc/a.txt
13     #NetScaler Commands
14       add lb vserver v2 http
15     #Shell Commands
16       echo "this is a 1" >> /etc/a.txt
17     #NetScaler Commands
18       add lb vserver v3 http
19     #end of file
20  kind: ConfigMap
```

```
21     metadata:
22         creationTimestamp: 2017-12-26T06:26:50Z
23         name: cpx-config
24         namespace: default
25         resourceVersion: "8865149"
26         selfLink: /api/v1/namespaces/default/configmaps/cpx-config
27         uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
```

You can specify the created ConfigMap, `cpx-config` in the YAML file used to deploy the NetScaler CPX instance as follows:

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: cpx-1
5    labels:
6      app: cpx-daemon
7    annotations:
8      NETSCALER_AS_APP: "True"
9  spec:
10   hostNetwork: true
11   containers:
12   - name: cpx
13     image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.1-33.47"
14     securityContext:
15       privileged: true
16     volumeMounts:
17     - name: config-volume
18       mountPath: /cpx/bootup_conf
19     env:
20     - name: "EULA"
21       value: "yes"
22     - name: "NS_NETMODE"
23       value: "HOST"
24     - name: "kubernetes_url"
25       value: "https://10.90.248.101:6443"
26     - name: "NS_MGMT_SERVER"
27       value: "10.90.248.99"
28     - name: "NS_MGMT_FINGER_PRINT"
29       value: "19:71:A3:36:85:0A:2B:62:24:65:0F:7E:72:CC:DC:AD:B8:BF
30         :53:1E"
31     - name: "NS_ROUTABLE"
32       value: "FALSE"
33     - name: "KUBERNETES_TASK_ID"
34       valueFrom:
35         fieldRef:
36           fieldPath: metadata.name
37     imagePullPolicy: Never
38   volumes:
39   - name: config-volume
40     configMap:
41       name: cpx-config
```

Once the NetScaler CPX instance is deployed and starts the configuration specified in the ConfigMap, `cpx-config` is applied to the NetScaler CPX instance.

Custom metrics configuration for NetScaler CPX using ConfigMap

September 7, 2026

You can provide custom metrics configuration JSON files to a NetScaler CPX™ instance in Kubernetes using ConfigMap. This allows you to add custom metrics schema files without overwriting the default metrics configuration files (`schema.json` and `reference_schema.json`) that include CPX.

During startup, the NetScaler CPX instance places the default metrics configuration files in the `/var/metrics_conf` directory. If you mount a ConfigMap at `/cpx/custom_metrics_conf`, CPX validates each JSON file and copies valid files into `/var/metrics_conf` alongside the defaults.

Notes:

- Custom JSON files must contain valid JSON. Invalid files are skipped during startup, and an error is logged in `boot.log`.
- Default files (`schema.json`, `reference_schema.json`) are always placed first.
- Custom files with the same name as a default file override the default file.

Configure custom metrics

You can configure custom metrics for NetScaler CPX by creating a ConfigMap and deploying the NetScaler CPX instance with the ConfigMap mounted.

To configure by using the CLI

Create the custom metrics configuration JSON files that you want to provide to the NetScaler CPX instance. The following is a sample custom configuration file:

```
1 {  
2  
3     "counter": "http_requests_total",  
4     "label": "method",  
5     "description": "Total HTTP requests by method"  
6 }
```

Create a ConfigMap from the files using the `kubectl create configmap` command. For example, to create a ConfigMap named `custom-metrics-conf` based on the configuration file `custom_schema.json`:

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

View the created ConfigMap using the `kubectl get configmap` command.

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

The following example shows sample output:

```
1 apiVersion: v1
2 data:
3   custom_schema.json: |
4     {
5
6       "counter": "http_requests_total",
7       "label": "method",
8       "description": "Total HTTP requests by method"
9     }
10
11 kind: ConfigMap
12 metadata:
13   creationTimestamp: 2026-05-14T06:26:50Z
14   name: custom-metrics-conf
15   namespace: default
16   resourceVersion: "8865149"
17   uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
```

To configure using a YAML file

Specify the created ConfigMap, `custom-metrics-conf`, in the YAML file used to deploy the NetScaler CPX instance. The following sample configuration shows how to include the ConfigMap in the deployment YAML:

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: cpx-1
5   labels:
6     app: cpx-daemon
7   annotations:
8     NETSCALER_AS_APP: "True"
9 spec:
10   containers:
11   - name: cpx
12     image: "quay.io/netScaler/netScaler-cpx:14.1-72.56"
13     securityContext:
14       privileged: true
15   volumeMounts:
16   - name: config-volume
17     mountPath: /cpx/bootup_conf
```

```
18   - name: custom-metrics-volume
19     mountPath: /cpx/custom_metrics_conf
20   env:
21     - name: "EULA"
22       value: "yes"
23     imagePullPolicy: IfNotPresent
24   volumes:
25     - name: config-volume
26       configMap:
27         name: cpx-config
28     - name: custom-metrics-volume
29       configMap:
30         name: custom-metrics-conf
```

Deploy the NetScaler CPX instance. Once the instance deploys and starts, the default metrics configuration files ([schema.json](#) and [reference_schema.json](#)) are placed in `/var/metrics_conf`. The custom JSON files from the ConfigMap mounted at `/cpx/custom_metrics_conf` are validated and copied into `/var/metrics_conf`.

Deploy NetScaler CPXs as Local DNS Caches for Kubernetes Nodes

September 7, 2026

Application pods in a Kubernetes cluster rely on DNS to communicate with other application pods. DNS requests from applications inside a Kubernetes cluster are handled by Kubernetes DNS (kube-dns). Due to wider adoption of microservices architectures, DNS request rates inside a Kubernetes cluster are increasing. As a result, Kubernetes DNS (kube-dns) is overburdened. Now you can deploy NetScaler CPX as a local DNS cache on each Kubernetes node and forward DNS requests from application pods in the node to NetScaler CPX. Hence, you can resolve DNS requests faster and significantly reduce the load on Kubernetes DNS.

To deploy NetScaler CPXs, a Kubernetes DaemonSet entity is used to schedule NetScaler CPX pods on each node in the Kubernetes cluster. A Kubernetes DaemonSet ensures that there is an instance of NetScaler CPX on each Kubernetes node in the cluster.

To make application pods direct traffic to CPX DNS pods, you need to create a Kubernetes service with endpoints as NetScaler CPX pods. Cluster IP of this service is used as the DNS endpoint for the application pods. To make sure that the application pods use NetScaler CPX service cluster IP address for DNS resolution, you need to update the kubelet configuration file on each node with NetScaler CPX service cluster IP.

The following environment variables are introduced to support the deployment of NetScaler CPX as NodeLocal DNS cache:

- **KUBE_DNS_SVC_IP**: Specifies the cluster IP address of the `kube-dns` service which is a mandatory argument to trigger the configuration on a NetScaler CPX pod. The NetScaler CPX pod directs DNS queries to this IP address when the DNS query response is not available in the NetScaler CPX cache.
- **CPX_DNS_SVC_IP**: Specifies the cluster IP address of the NetScaler CPX service. The `CPX_DNS_SVC_IP` environment variable is used to configure local DNS on nodes. When you configure this variable, an `iptables` rule is added to direct the DNS requests originating from application pods to the local NetScaler CPX pod inside the node.
- **NS_DNS_FORCE_TCP**: This environment variable forces using TCP for DNS requests even if the queries are received over UDP.
- **NS_DNS_EXT_RESOLV_IP**: Specifies the IP address of the external name server to direct the DNS requests for a specific domain.
- **NS_DNS_MATCH_DOMAIN**: Specifies the external domain string to be matched against to direct the queries to the external name server.

Deploy NetScaler CPXs as DNS Caches on Nodes

Deploying NetScaler CPX as local DNS cache for a Kubernetes cluster includes the following tasks:

On the master node:

- Create a Kubernetes service with endpoints as NetScaler CPX pods
- Create a ConfigMap for defining environment variables for NetScaler CPX pods
- Schedule NetScaler CPX pods on each node in the Kubernetes cluster using a Kubernetes DaemonSet.

On worker nodes:

- Modify the kubelet configuration file with the cluster IP address of NetScaler CPX service to forward DNS requests to NetScaler CPXs.

Configuration on the Kubernetes Master Node

Perform the following steps on the Kubernetes master node to deploy NetScaler CPX as the local DNS cache for nodes:

1. Create a service with NetScaler CPX pods as endpoints using the `cpx_dns_svc.yaml` file.

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

The `cpx_dns_svc.yaml` file is provided as follows:

```
1      apiVersion: v1
2      kind: Service
3      metadata:
4        name: cpx-dns-svc
5        labels:
6          app: cpxd
7      spec:
8        ports:
9          - protocol: UDP
10            port: 53
11            name: dns
12          - protocol: TCP
13            port: 53
14            name: dns-tcp
15        selector:
16          app: cpx-daemon
```

2. Get the IP address of the NetScaler CPX service.

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

3. Get the IP address of the Kube DNS service.

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

4. Create a ConfigMap for defining environment variables for NetScaler CPX pods. These environment variables are used to pass IP addresses of NetScaler CPX service and Kube DNS service. In this step, a sample ConfigMap `cpx-dns-cache` is created using the environment variables specified as data (key-value pairs) in a file.

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

The following is a sample file with the environment variables as key-value pairs.

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

The following is a sample ConfigMap:

```
1 apiVersion: v1
2 data:
3   CPX_DNS_SVC_IP: 10.111.95.145
4   EULA: "yes"
5   KUBE_DNS_SVC_IP: 10.96.0.10
6   NS_CPX_LITE: "1"
```

```
7   NS_DNS_EXT_RESLV_IP: 10.102.217.142
8   NS_DNS_MATCH_DOMAIN: citrix.com
9   PLATFORM: CP1000
10  kind: ConfigMap
11  metadata:
12    creationTimestamp: "2019-10-15T07:45:54Z"
13    name: cpx-dns-cache
14    namespace: default
15    resourceVersion: "8026537"
16    selfLink: /api/v1/namespaces/default/configmaps/cpx-dns-cache
17    uid: 8d06f6ee-133b-4e1a-913c-9963cbf4f48
```

5. Create a Kubernetes DaemonSet for NetScaler CPX on the master node.

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

The `cpx_daemonset.yaml` file is provided as follows:

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

```
36     path: /proc/1/ns
37     type: Directory
```

Configuration on Worker Nodes in the Kubernetes Cluster

Once you complete configuration on master node, perform the following step on worker nodes:

1. Modify the kubelet configuration file so that application pods can use NetScaler CPX service cluster IP for DNS resolution using one of the following steps:
 - Follow the steps in [reconfigure a Node's kubelet](#) and modify the `--cluster-dns` argument value in the following format.

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

or

- Edit the `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` file and modify the `--cluster-dns` argument using the following steps.
 - a) Edit the kubelet configuration and specify the cluster IP address of NetScaler CPX service and `kube-dns` service IP address for the `--cluster-dns` argument.

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

- b) Reload the kubelet of nodes using the following commands:

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

Deploy NetScaler CPX Proxy on Google Compute Engine

September 7, 2026

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

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

CPX and CPX Express are the same images. When you license and install the CPX image using NetScaler Console, 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 NetScaler CPX
- Docker open source available from GCE
- NetScaler Console running on premises with internet or VPN connection to GCE

Note

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

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 NetScaler CPX image.
4. Create a CPX instance.
5. License NetScaler CPX through NetScaler Console.
6. Configure Load Balancing Services on NetScaler CPX and verify the configuration.
 - a) Install NGINX web servers.
 - b) Configure NetScaler 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:

```

1 $ sudo curl -ssl https://get.docker.com/ | sh
2 % Total % Received % Xferd Average Speed Time Time Time Current
3 Dload Upload Total Spent Left Speed
4 0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve
   host: xn--ssl-1n0a
5 100 17409 100 17409 0 0 21510 0 --:--:-- --:--:-- --:--:-- 21492
6 apparmor is enabled in the kernel and apparmor utils were already
   installed
7 \+ sudo -E sh -c apt-key add -
8 \+ echo -----BEGIN PGP PUBLIC KEY BLOCK-----
9 Version: GnuPG v1
10
11 mQINBFWln24BEADrBl5p99uKh8+rpvqJ48u4eTtjeXAWbslJotmC/CakbNSqOb9o
12 ddfzRvGveJVERT/Q/mlvEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
13 mYdoe/BJ/J/6rH0jq70mk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNSBJXoCvPn
14 TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyU/jPpWFK28TR8yFVlzYFwibj5WK
15 dHM7ZTqlA1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccy0i+ULU0z8mHUJ3iEMirpT
16 X+80KaN/ZjibfsB0CjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
17 HOoyC6qVeOgKrRiKxzymj0FIMLru/iFF5pSWcBQB7PYlt8J0G80lAcPr6VCiN+4c
18 NKv03SdvA69dCOj79Pu09IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHMH1fDeCqkZ
19 hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Zaltn0svoGs+wX3Z/H6/8P5anog43U
20 65c0A+64Jj00rNDR8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+cOMHUU4HA7iM
21 zHrouAdYeTZeZEQA7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFncTKdv+rwARAQAB
22 tDdEb2NrZXIgmUmVsZWZzZSBub29sICHyZWxlyXNlZG9ja2VyKSA8ZG9ja2VyQGRv
23 Y2tldi5jb20+iQICBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4Syo0CV
24 cnybl18vzwBEcp4VCRbXvHvOXty1gccVIV8/aJqNKgBV97lY3vrp0yiIeB8ETQeg
25 srxFE7t/Gz0rsLObqfLEHdmn5iBJRkhlCpzej0nyB3Z0IJB6Uog0/msQVYe5CXJ
26 l6uwr0AmoicBLrVLDaktxVh9RWch0l0KZR2FpHu8h+uM0/zySqIdlYfLa3y5oH
27 scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdcAHR+bXhA/RW8QHeeMFlIwW
28 7Z2jYHyuHmDnWG2yUrnCqAJTrWV+OfKRIzzJFBs4e88ru5h2ZIXdRepw/+COYj34
29 LyzxR2cxr2u/xvxwXCkSM7F4KZaphD+1ws61FhnUMi/PERMYfTFuvPrCkq4gyBj
30 t3fFpZ2NR/fKW87Q0eVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsrxkFQ2jNp
31 D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaH0
32 W1zblWy/CFr6VFl963jrjJgag0G6tNtBZLrclZgWh0QpeZZ5Lbvz2ZA5CqRrFAVc
33 wPNW1f0bFIRtqV6vuVluFOPCMAAnOnqr02w9t17iVQj03oVN0mbQi9vjuExXh1Yo
34 ScVeti06LSmlQfVEVRTqHLMgXyR/EMo7iQICBBABCgAGBQJXSWBLAAoJEFyzYeVS
35 \+w0QeH0QAI6btAfYwYPuAjrUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+l/R08nu0d
36 sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
37 NydGFzmeOKbAlrxIMECnSTG/TkFVY09Ntlv9vSN2BupmTagTRErXLZKnVsWRzp+X
38
39 \-----END PGP PUBLIC KEY BLOCK-----
40
41 OK
42 \+ sudo -E sh -c mkdir -p /etc/apt/sources.list.d
43 \+ dpkg --print-architecture
44 \+ sudo -E sh -c echo deb \[\[arch=amd64\]\] https://apt.dockerproject.
   org/repo ubuntu-yakkety main \> /etc/apt/sources.list.d/docker.list
45 \+ sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-
   engine
46 Hit:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety InRelease
47 Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates
   InRelease \[102 kB\]
48 Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports

```

```
InRelease \[102 kB\]
49 Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted
    Sources \[5,376 B\]
50 Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse
    Sources \[181 kB\]
51 Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
    Sources \[8,044 kB\]
52 Get:7 http://archive.canonical.com/ubuntu yakkety InRelease \[11.5 kB\]
53 Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease
    \[102 kB\]
54 Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease
    \[47.3 kB\]
55 Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main
    Sources \[903 kB\]
56 Get:11 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    restricted Sources \[2,688 B\]
57 Get:12 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    universe Sources \[57.9 kB\]
58 Get:13 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    multiverse Sources \[3,172 B\]
59 Get:14 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main Sources \[107 kB\]
60 Get:15 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main amd64 Packages \[268 kB\]
61 Get:16 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    main Translation-en \[122 kB\]
62 Get:17 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    universe amd64 Packages \[164 kB\]
63 Get:18 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    universe Translation-en \[92.4 kB\]
64 Get:19 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    multiverse amd64 Packages \[4,840 B\]
65 Get:20 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
    multiverse Translation-en \[2,708 B\]
66 Get:21 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe Sources \[2,468 B\]
67 Get:22 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    main Sources \[2,480 B\]
68 Get:23 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    main amd64 Packages \[3,500 B\]
69 Get:24 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe amd64 Packages \[3,820 B\]
70 Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
    universe Translation-en \[1,592 B\]
71 Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64
    Packages \[2,480 B\]
72 Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources
    \[47.7 kB\]
73 Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
    Packages \[2,453 B\]
74 Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe
    Sources \[20.7 kB\]
75 Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
```

```
Sources \[1,140 B\]
76 Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted
  Sources \[2,292 B\]
77 Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64
  Packages \[150 kB\]
78 Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main
  Translation-en \[68.0 kB\]
79 Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe
  amd64 Packages \[77.2 kB\]
80 Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe
  Translation-en \[47.3 kB\]
81 Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
  amd64 Packages \[2,832 B\]
82 Fetched 10.8 MB in 2s (4,206 kB/s)
83 Reading package lists... Done
84 Reading package lists...
85 Building dependency tree...
86 Reading state information...
87 The following additional packages will be installed:
88 aufs-tools cgroupfs-mount libltdl7
89 The following NEW packages will be installed:
90 aufs-tools cgroupfs-mount docker-engine libltdl7
91 0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.
92 Need to get 21.2 MB of archives.
93 After this operation, 111 MB of additional disk space will be used.
94 Get:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
  amd64 aufs-tools amd64 1:3.2+20130722-1.1ubuntu1 \[92.9 kB\]
95 Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
  amd64 cgroupfs-mount all 1.3 \[5,778 B\]
96 Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main amd64
  libltdl7 amd64 2.4.6-1 \[38.6 kB\]
97 Get:4 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
  docker-engine amd64 17.05.0~ce-0~ubuntu-yakkety \[21.1 MB\]
98 Fetched 21.2 MB in 1s (19.8 MB/s)
99 Selecting previously unselected package aufs-tools.
100 (Reading database ... 63593 files and directories currently installed.)
101 Preparing to unpack .../aufs-tools\_1%3a3.2+20130722-1.1ubuntu1\_amd64.
  deb ...
102 Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
103 Selecting previously unselected package cgroupfs-mount.
104 Preparing to unpack .../cgroupfs-mount\_1.3\_all.deb ...
105 Unpacking cgroupfs-mount (1.3) ...
106 Selecting previously unselected package libltdl7:amd64.
107 Preparing to unpack .../libltdl7\_2.4.6-1\_amd64.deb ...
108 Unpacking libltdl7:amd64 (2.4.6-1) ...
109 Selecting previously unselected package docker-engine.
110 Preparing to unpack .../docker-engine\_17.05.0~ce-0~ubuntu-yakkety\_
  _amd64.deb ...
111 Unpacking docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
112 Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
113 Processing triggers for ureadahead (0.100.0-19) ...
114 Setting up cgroupfs-mount (1.3) ...
115 Processing triggers for libc-bin (2.24-3ubuntu2) ...
```

```
116 Processing triggers for systemd (231-9ubuntu4) ...
117 Setting up libltdl7:amd64 (2.4.6-1) ...
118 Processing triggers for man-db (2.7.5-1) ...
119 Setting up docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
120 Created symlink /etc/systemd/system/multi-user.target.wants/docker.
    service → /lib/systemd/system/docker.service.
121 Created symlink /etc/systemd/system/sockets.target.wants/docker.socket
    → /lib/systemd/system/docker.socket.
122 Processing triggers for ureadahead (0.100.0-19) ...
123 Processing triggers for libc-bin (2.24-3ubuntu2) ...
124 Processing triggers for systemd (231-9ubuntu4) ...
125 \+ sudo -E sh -c docker version
126 Client:
127 Version: 17.05.0-ce
128 API version: 1.29
129 Go version: go1.7.5
130 Git commit: 89658be
131 Built: Thu May 4 22:15:36 2017
132 OS/Arch: linux/amd64
133
134 Server:
135 Version: 17.05.0-ce
136 API version: 1.29 (minimum version 1.12)
137 Go version: go1.7.5
138 Git commit: 89658be
139 Built: Thu May 4 22:15:36 2017
140 OS/Arch: linux/amd64
141 Experimental: false
142
143 If you would like to use Docker as a non-root user, you should now
    consider
144 adding your user to the "docker" group with something like:
145
146 sudo usermod -aG docker albert\_lee
147
148 Remember that you will have to log out and back in for this to take
    effect.
149
150 WARNING: Adding a user to the "docker" group will grant the ability to
    run
151 containers which can be used to obtain root privileges on the
152 docker host.
153 Refer to https://docs.docker.com/engine/security/security/#docker-
    daemon-attack-surface
154 for more information.
155
156 $
157
158 **$ sudo docker info**
159 Containers: 0
160 Running: 0
161 Paused: 0
162 Stopped: 0
```

```
163 Images: 0
164 Server Version: 17.05.0-ce
165 Storage Driver: aufs
166 Root Dir: /var/lib/docker/aufs
167 Backing Filesystem: extfs
168 Dirs: 0
169 Dirperm1 Supported: true
170 Logging Driver: json-file
171 Cgroup Driver: cgroupfs
172 Plugins:
173 Volume: local
174 Network: bridge host macvlan null overlay
175 Swarm: inactive
176 Runtimes: runc
177 Default Runtime: runc
178 Init Binary: docker-init
179 containerd version: 9048e5e50717ea4497b757314bad98ea3763c145
180 runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228
181 init version: 949e6fa
182 Security Options:
183 apparmor
184 seccomp
185 Profile: default
186 Kernel Version: 4.8.0-51-generic
187 Operating System: Ubuntu 16.10
188 OSType: linux
189 Architecture: x86\_64
190 CPUs: 1
191 Total Memory: 3.613GiB
192 Name: docker-7
193 ID: R5TW:VKXK:EKGR:GHWM:UNU4:LPJH:IQY5:X77G:NNRQ:HWBY:LIUD:4ELQ
194 Docker Root Dir: /var/lib/docker
195 Debug Mode (client): false
196 Debug Mode (server): false
197 Registry: https://index.docker.io/v1/
198 Experimental: false
199 Insecure Registries:
200 127.0.0.0/8
201 Live Restore Enabled: false
202
203 WARNING: No swap limit support
204 $
205
206 **$ sudo docker images**
207 REPOSITORY TAG IMAGE ID CREATED SIZE
208 $
209
210 **$ sudo docker ps**
211 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
212 $
```

Step 2: Configure Remote API Communication with the Docker Instance

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

```

1
2  **cd /etc/systemd/system**
3  **sudo vi docker-tcp.socket**
4  **cat docker-tcp.socket**
5  \[Unit\]
6  **Description=Docker Socket for the API
7  \[Socket\]
8  ListenStream=4243
9  BindIPv6Only=both
10 Service=docker.service
11 \[Install\]
12 WantedBy=sockets.target**
13
14 $ **sudo systemctl enable docker-tcp.socket**
15 Created symlink /etc/systemd/system/sockets.target.wants/docker-tcp.
    socket → /etc/systemd/system/docker-tcp.socket.
16 **sudo systemctl enable docker.socket**
17 **sudo systemctl stop docker**
18 **sudo systemctl start docker-tcp.socket**
19 **sudo systemctl start docker**
20 $ **sudo systemctl status docker**
21 • docker.service - Docker Application Container Engine
22 Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
    preset: enabled)
23 Active: **active (running)** since Wed 2017-05-31 12:52:17 UTC; 2s ago
24 Docs: https://docs.docker.com
25 Main PID: 4133 (dockerd)
26 Tasks: 16 (limit: 4915)
27 Memory: 30.1M
28 CPU: 184ms
29 CGroup: /system.slice/docker.service
30 └─4133 /usr/bin/dockerd -H fd://
31 └─4137 docker-containerd -l unix:///var/run/docker/libcontainerd/docker
    -containerd.sock --metrics-interval=0 --start-timeout 2m -
32
33 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
    :52:17.300890402Z" level=warning msg="Your kernel does not support
    cgroup rt peri
34 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
    :52:17.301079754Z" level=warning msg="Your kernel does not support
    cgroup rt runt
35 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
    :52:17.301681794Z" level=info msg="Loading containers: start."
36 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
    :52:17.417539064Z" level=info msg="Default bridge (docker0) is
    assigned with an I
37 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12

```

```

:52:17.465011600Z" level=info msg="Loading containers: done."
38 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
:52:17.484747909Z" level=info msg="Daemon has completed
initialization"
39 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
:52:17.485119478Z" level=info msg="Docker daemon" commit=89658be
graphdriver=aufs
40 May 31 12:52:17 docker-7 systemd\[1\]: Started Docker Application
Container Engine.
41 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
:52:17.503832254Z" level=info msg="API listen on /var/run/docker.
sock"
42 May 31 12:52:17 docker-7 dockerd\[4133\]: time="2017-05-31T12
:52:17.504061522Z" level=info msg="API listen on \[::\]:4243"
43 $
44
45 (external)$ **curl 104.199.209.157:4243/version**
46 {
47   "Version":"17.05.0-ce","ApiVersion":"1.29","MinAPIVersion":"1.12","
GitCommit":"89658be","GoVersion":"go1.7.5","Os":"linux","Arch":"
amd64","KernelVersion":"4.8.0-52-generic","BuildTime":"2017-05-04
T22:15:36.071254972+00:00" }
48
49 (external)$

```

Step 3: Install NetScaler CPX Image

Get the NetScaler CPX image from Docker App Store. The CPX Express and CPX have the same image. But, when you license and install the CPX image using NetScaler Console 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.

```

1 $ **sudo docker pull store/citrix/citrixadccpx:13.0-36.29**
2 13.0-36.29: Pulling from store/citrix/citrixadccpx
3 4e1f679e8ab4: Pull complete
4 a3ed95caeb02: Pull complete
5 2931a926d44b: Pull complete
6 362cd40c5745: Pull complete
7 d10118725a7a: Pull complete
8 1e570419a7e5: Pull complete
9 d19e06114233: Pull complete
10 d3230f008ffd: Pull complete
11 22bdb10a70ec: Pull complete
12 1a5183d7324d: Pull complete
13 241868d4ebff: Pull complete
14 3f963e7ae2fc: Pull complete
15 fd254cf1ea7c: Pull complete
16 33689c749176: Pull complete
17 59c27bad28f5: Pull complete
18 588f5003e10f: Pull complete

```

```

19 Digest: sha256:31
    a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe
20 Status: Downloaded newer image for store/citrix/citrixadccpx:13.0-36.29
21 $
22
23 $ **sudo docker images**
24 REPOSITORY TAG IMAGE ID CREATED SIZE
25 store/citrix/citrixadccpx:13.0-36.29 6fa57c38803f 3 weeks ago 415MB
26 $

```

Step 4: Create a NetScaler CPX Instance

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

```

1 bash-2.05b\# **CHOST=${
2 1:-localhost }
3 **
4 bash-2.05b\# **echo | openssl s_client -connect $CHOST:443 | openssl
    x509 -fingerprint -noout | cut -d'=' -f2**
5 depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU =
    Internal, CN = Test Only Cert
6 verify error:num=18:self signed certificate
7 verify return:1
8 depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU =
    Internal, CN = Test Only Cert
9 verify return:1
10 DONE
11 24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4
12 bash-2.05b\#
13
14 $ **sudo docker run -dt -p 50000:88 -p 5080:80 -p 5022:22 -p 5443:443 -
    p 5163:161/udp -e NS\_HTTP\_PORT=5080 -e NS\_HTTPS\_PORT=5443 -e NS\
    \_SSH\_PORT=5022 -e NS\_SNMP\_PORT=5163 -e EULA=yes -e LS\_IP=xx.xx.
    xx.xx -e PLATFORM=CP1000 --privileged=true --ulimit core=-1 -e NS\
    \_MGMT\_SERVER=xx.xx.xx.xx:xxxx -e NS\_MGMT\_FINGER\_PRINT=24:AA:8B
    :91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4 --env NS\
    \_ROUTABLE=false --env HOST=104.199.209.157 store/citrix/citrixadccpx
    :13.0-36.29**
15 44calc6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
16 $
17
18 $ **sudo docker ps**
19 CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
20 44calc6c0907 store/citrix/citrixadccpx:13.0-36.29 "/bin/sh -c 'bash ...
    " 19 seconds ago Up 17 seconds 0.0.0.0:5022->22/tcp,
    0.0.0.0:5080->80/tcp, 0.0.0.0:50000->88/tcp, 0.0.0.0:5163->161/
    udp, 0.0.0.0:5443->443/tcp gifted\_perlman
21 $
22
23 $ **ssh -p 5022 root@localhost**

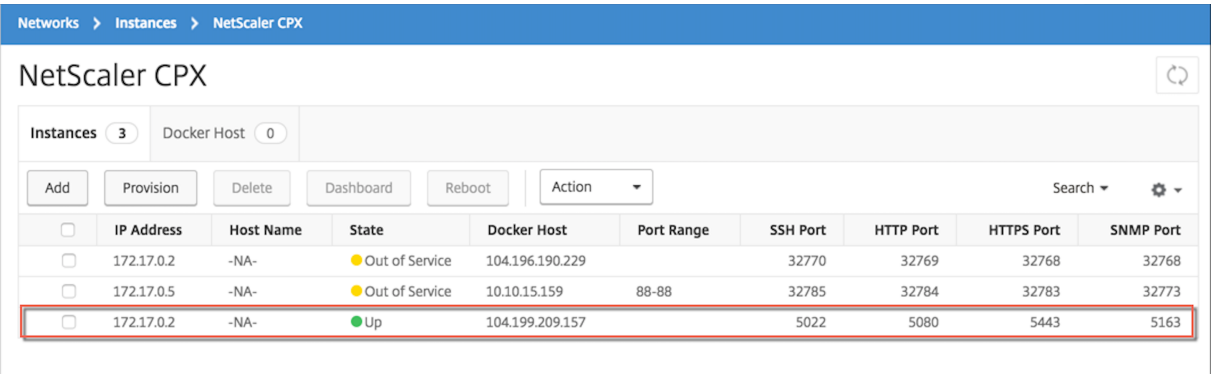
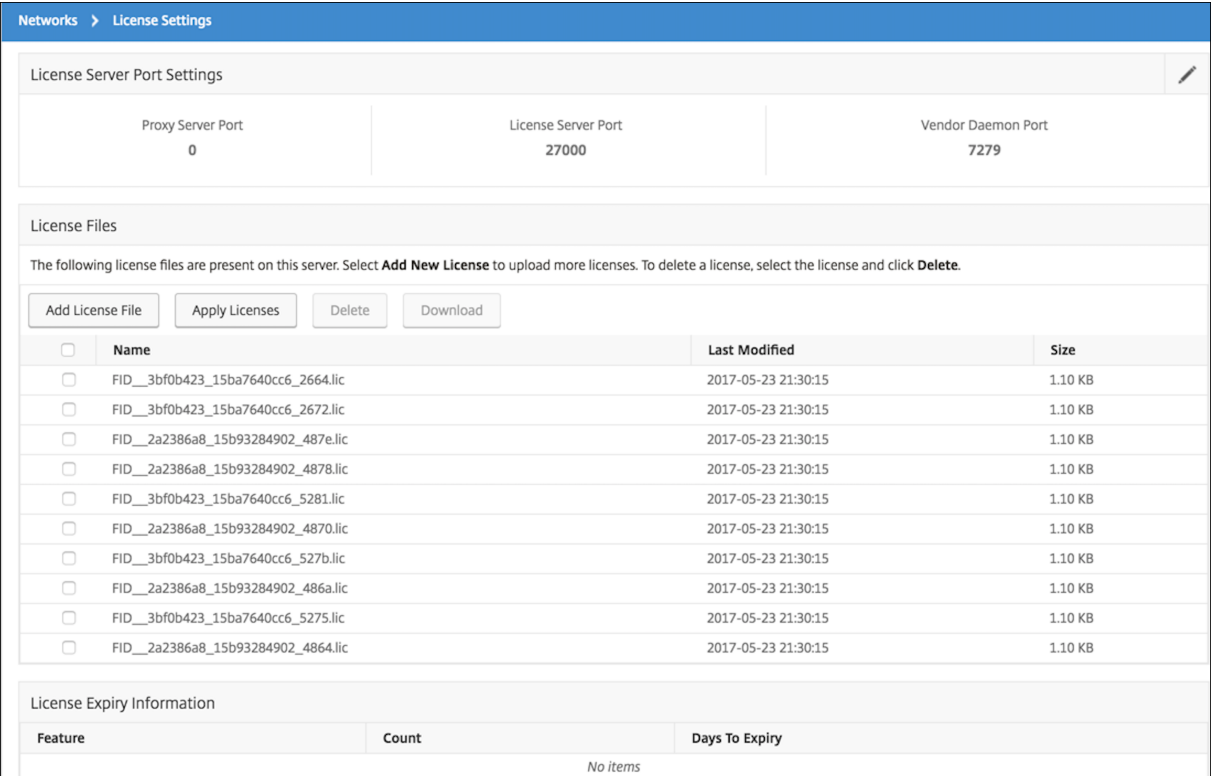
```

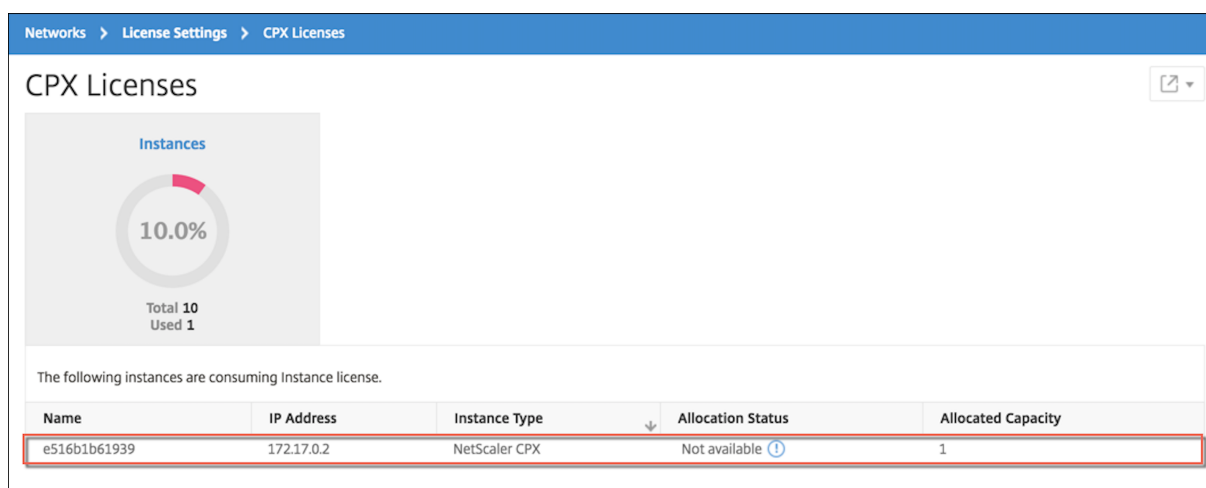
```
24 root@localhost's password:
25 Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)
26
27 \* Documentation: https://www.citrix.com/
28 Last login: Mon Jun 5 18:58:51 2017 from xx.xx.xx.xx
29 root@44c1c6c0907:~\#
30 root@44c1c6c0907:~\#
31 root@44c1c6c0907:~\# **cli\_script.sh 'show ns ip'**
32 exec: show ns ip
33 Ipaddress Traffic Domain Type Mode Arp Icmp Vserver State
34 \-----
35 1\) 172.17.0.2 0 NetScaler IP Active Enabled Enabled NA Enabled
36 2\) 192.0.0.1 0 SNIP Active Enabled Enabled NA Enabled
37 Done
38 root@44c1c6c0907:~\# **cli\_script.sh 'show licenseserver'**
39 exec: show licenseserver
40 1\) ServerName: xx.xx.xx.xxPort: 27000 Status: 1 Grace: 0 Gptimeleft: 0
41 Done
42 root@44c1c6c0907:~\# cli\_script.sh 'show capacity'
43 exec: show capacity
44 Actualbandwidth: 1000 Platform: CP1000 Unit: Mbps Maxbandwidth: 3000
   Minbandwidth: 20 Instancecount: 0
45 Done
46 root@44c1c6c0907:~\#
47
48 $ **sudo iptables -t nat -L -n**
49 Chain PREROUTING (policy ACCEPT)
50 target prot opt source destination
51 DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
52
53 Chain INPUT (policy ACCEPT)
54 target prot opt source destination
55
56 Chain OUTPUT (policy ACCEPT)
57 target prot opt source destination
58 DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
59
60 Chain POSTROUTING (policy ACCEPT)
61 target prot opt source destination
62 MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
63 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
64 MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
65 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
66 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
67 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
68
69 Chain DOCKER (2 references)
70 target prot opt source destination
71 RETURN all -- 0.0.0.0/0 0.0.0.0/0
72 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
73 DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
74 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
75 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
```

```
76 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
77 $
```

Step 5: License NetScaler CPX Through NetScaler Console

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





Step 6: Configure Load Balancing Services on NetScaler CPX and Verify the Configuration

First, install NGINX web servers on the Docker host. Then, configure load balancing on NetScaler CPX to load balance the two web servers, and test the configuration.

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

```

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

```

```

26 DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
27
28 Chain POSTROUTING (policy ACCEPT)
29 target prot opt source destination
30 MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
31 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
32 MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
33 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
34 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
35 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
36 MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80
37 MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80
38
39 Chain DOCKER (2 references)
40 target prot opt source destination
41 RETURN all -- 0.0.0.0/0 0.0.0.0/0
42 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
43 DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
44 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
45 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
46 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
47 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80
48 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80
49 $

```

Configure NetScaler CPX for Load Balancing and Verify Distribution of Load to both Web Services

```

1 $ **ssh -p 5022 root@localhost**
2 root@localhost's password:
3 Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)
4
5 \* Documentation: https://www.citrix.com/
6 Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1
7 root@44calc6c0907:~\#
8 root@44calc6c0907:~\#
9 root@44calc6c0907:~\#
10 root@44calc6c0907:~\#
11 root@44calc6c0907:~\# **cli\_script.sh "add service web1 172.17.0.3
    HTTP 80"**
12 exec: add service web1 172.17.0.3 HTTP 80
13 Done
14 root@44calc6c0907:~\# **cli\_script.sh "add service web2 172.17.0.4
    HTTP 80"**
15 exec: add service web2 172.17.0.4 HTTP 80
16 Done
17 root@44calc6c0907:~\# **cli\_script.sh "add lb vserver cpx-vip HTTP
    172.17.0.2 88"**
18 exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
19 Done
20 root@44calc6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web1"**
21 exec: bind lb vserver cpx-vip web1
22 Done
23 root@44calc6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web2"**

```

```
24 exec: bind lb vserver cpx-vip web2
25 Done
26 root@44ca1c6c0907:~\#
27
28 root@44ca1c6c0907:~\# **cli\_script.sh 'show lb vserver cpx-vip'**
29 exec: show lb vserver cpx-vip
30
31 cpx-vip (172.17.0.2:88) - HTTP Type: ADDRESS
32 State: UP
33 Last state change was at Mon Jun 5 19:01:49 2017
34 Time since last state change: 0 days, 00:00:42.620
35 Effective State: UP
36 Client Idle Timeout: 180 sec
37 Down state flush: ENABLED
38 Disable Primary Vserver On Down : DISABLED
39 Appflow logging: ENABLED
40 Port Rewrite : DISABLED
41 No. of Bound Services : 2 (Total) 2 (Active)
42 Configured Method: LEASTCONNECTION
43 Current Method: Round Robin, Reason: A new service is bound
    BackupMethod: ROUNDROBIN
44 Mode: IP
45 Persistence: NONE
46 Vserver IP and Port insertion: OFF
47 Push: DISABLED Push VServer:
48 Push Multi Clients: NO
49 Push Label Rule: none
50 L2Conn: OFF
51 Skip Persistency: None
52 Listen Policy: NONE
53 IcmpResponse: PASSIVE
54 RHistate: PASSIVE
55 New Service Startup Request Rate: 0 PER\_SECOND, Increment Interval: 0
56 Mac mode Retain Vlan: DISABLED
57 DBS\_LB: DISABLED
58 Process Local: DISABLED
59 Traffic Domain: 0
60 TROFS Persistence honored: ENABLED
61 Retain Connections on Cluster: NO
62
63 2\) web1 (172.17.0.3: 80) - HTTP State: UP Weight: 1
64 3\) web2 (172.17.0.4: 80) - HTTP State: UP Weight: 1
65 Done
66 root@44ca1c6c0907:~\#
67
68 (external)$ **curl 104.199.209.157:50000**
69 \<\!DOCTYPE html\>
70 \<html\>
71 \<head\>
72 \<title\>Welcome to nginx!\</title\>
73 \<style\>
74 body {
75
```

```

76 width: 35em;
77 margin: 0 auto;
78 font-family: Tahoma, Verdana, Arial, sans-serif;
79 }
80
81 \</style\>
82 \</head\>
83 \<body\>
84 \<h1\>Welcome to nginx\!\</h1\>
85 \<p\>If you see this page, the nginx web server is successfully
    installed and
86 working. Further configuration is required.\</p\>
87
88 \<p\>For online documentation and support please refer to
89 \<a href="http://nginx.org/"\>nginx.org\</a\>.\<br/\>
90 Commercial support is available at
91 \<a href="http://nginx.com/"\>nginx.com\</a\>.\</p\>
92
93 \<p\>\<em\>Thank you for using nginx.\</em\>\</p\>
94 \</body\>
95 \</html\>
96 (external)$
97
98
99 (external)$ for i in {
100 1..100 }
101 ; **do curl http://104.199.209.157:50000 -o /dev/null ; done**
102
103 % Total      % Received % Xferd  Average Speed   Time    Time       Time
104      Current
105
106      Speed
107 100    612    100    612      0      0   1767      0  --:--:--  --:--:--
108      --:--:--  1768
109
110 % Total      % Received % Xferd  Average Speed   Time    Time       Time
111      Current
112
113      Speed
114 100    612    100    612      0      0   1893      0  --:--:--  --:--:--
115      --:--:--  1894
116
117 % Total      % Received % Xferd  Average Speed   Time    Time       Time
118      Current
119
120      Speed
121 100    612    100    612      0      0   1884      0  --:--:--  --:--:--

```

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

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

```
192
193
194
195 root@44ca1c6c0907:~\# **cli\_script.sh 'stat lb vserver cpx-vip'**
196
197 exec: stat lb vserver cpx-vip
198
199
200
201 Virtual Server Summary
202
203           vsvrIP  port  Protocol  State  Health
204   actSvcs
205 cpx-vip      172.17.0.2   88      HTTP      UP      100
206           2
207
208   inactSvcs
209 cpx-vip      0
210
211
212
213
214 Virtual Server Statistics
215
216           Rate (/s)
217   Total
218
219 Vserver hits      0
220   101
221
222 Requests      0
223   101
224
225 Responses      0
226   101
227
228 Request bytes      0
229   8585
230
231 Response bytes      0
232   85850
233
234 Total Packets rcvd      0
235   708
236
237 Total Packets sent      0
238   408
239
240 Current client connections  --
241   0
```

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

```
270      0
271 web1      172.17.0.3    80      HTTP      UP      51
272      0/s
273 web2      172.17.0.4    80      HTTP      UP      50
274      0/s
275 Done
276
277 root@44ca1c6c0907:~\#
```

NetScaler CPX troubleshooting

September 7, 2026

This document explains how to troubleshoot issues that you may encounter while using NetScaler CPX. 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 NetScaler CPX.

- Why is the memory usage percentage always high in NetScaler CPX?

The memory usage percentage is determined by comparing the current memory usage of NetScaler CPX to the total memory NetScaler CPX allocates itself from the operating system (OS). NetScaler CPX starts by allocating a minimum necessary memory from the OS for booting and basic operations. Initially, because NetScaler CPX allocates the minimum necessary memory for its operations, the memory consumption percentage is high. Eventually, as the need for more resources grows with the scaling of NetScaler CPX operations, NetScaler CPX incrementally allocates additional memory from the OS and the memory usage is also high, resulting in high memory usage percentage.

Thus, due to NetScaler CPX's dynamic memory allocation strategy, NetScaler CPX typically shows a high memory usage percentage, reflecting its efficient use of resources rather than a lack of available memory.

- How can I view NetScaler CPX logs?

You can view NetScaler CPX logs using the `kubectl logs` command if NetScaler CPX is deployed with the `tty:true` option. You can run the following command to display the logs:

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

Example,

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

The following is an example of the NetScaler CPX pod deployment with the `tty: true` option:

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

You can find more boot logs in the `/cpx/log/boot.log` file of NetScaler CPX 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 NetScaler CPX?

You can run the following command on the shell interface of the Kubernetes master node to collect the NetScaler CPX technical support bundle:

```
1   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 NetScaler CPX's file system. Use `scp` or `kubectl cp` to copy the technical support bundle from NetScaler CPX to the desired destination.

Example:

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

- Why is NetScaler CPX 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:

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

Example:

```
1 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 NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX pod and the Kubernetes master node:

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

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

Example:

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

scp: You can use the command to copy files between the NetScaler CPX 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:

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

Example:

```
1 root@localhost:~# scp nsroot@192.168.29.198:/cpx/log/boot.log /
  tmp/cpx-boot.log
2 nsroot@192.168.29.198's password:
3 boot.log
4 100% 7880      5.1MB/s   00:00
5 root@localhost:~#
```

- How do I capture packets on NetScaler CPX?

To capture packets on NetScaler CPX, launch the shell interface of NetScaler CPX using the `kubectl exec` command. Run the following command to launch the shell interface of the NetScaler CPX pod:

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

Example:

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

And, run the following command to begin packet capture:

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

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

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

You can view the packets captured in a *.cap* file in the */cpx/nstrace/time-stamp* directory on the NetScaler CPX file system.

- Why is the license server not configured even when NetScaler CPX is deployed with the `LS_IP=<ADM-IP>` environment variable?

Ensure that the license server is accessible from the node on which NetScaler CPX is deployed. You can use the `ping <ADM-IP>` command to verify the connectivity from the NetScaler CPX node to NetScaler Console.

If NetScaler Console 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 NetScaler CPX pod:

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

Example:

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

- Why is pooled license not configured on NetScaler CPX even after a successful license server configuration on NetScaler CPX?

Verify the license configuration logs in the */cpx/log/boot.log* file. You can also verify the configured pooled license on NetScaler CPX using the following command on the shell interface of the NetScaler CPX pod:

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

Example,

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

```

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

```

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 NetScaler CPX by using the following command. Run the command on the shell interface of NetScaler CPX pod:

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

Example:

```

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

```

- Why do NITRO API calls get *Connection Refused* response from NetScaler CPX?

The default port for NITRO APIs is 9080 (unsecure) and 9443 (secure) from the NetScaler CPX release 12.1 onwards. Ensure that the NITRO port of NetScaler CPX 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 NetScaler CPX container in the NetScaler CPX container section:

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

Example:

```

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

```

- Why does the NSPPE process in NetScaler CPX consume most of the CPU usage even when there is no or little traffic?

If NetScaler CPX 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 NetScaler CPX 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 NetScaler CPX not listed in NetScaler Console even when it was deployed with the required environment variables for registration with NetScaler Console?

You can find the logs for NetScaler CPX registration with NetScaler Console in the `/cpx/log/-boot.log` file on the NetScaler CPX file system.

You can verify the accessibility of the NetScaler Console IP address from the NetScaler CPX pod using the `ping` command. Also, ensure that all the required environment variables for NetScaler Console registration are configured for the NetScaler CPX container.

- `NS_MGMT_SERVER`: Specifies the ADM-IP address or FQDN.
 - `HOST`: Specifies the node IP address.
 - `NS_HTTP_PORT`: Specifies the mapped HTTP- port on node.
 - `NS_HTTPS_PORT`: Specifies the mapped HTTPS port on node.
 - `NS_SSH_PORT`: Specifies the mapped SSH port on node.
 - `NS_SNMP_PORT`: Specifies the mapped SNMP port on node.
 - `NS_ROUTABLE`: NetScaler CPX pod IP address is not routable from outside.
 - `NS_MGMT_USER`: Specifies the ADM username.
 - `NS_MGMT_PASS`: Specifies the ADM password.
- 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 NetScaler CPX. 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:

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

Example:

```
1 root@087a1e34642d:/# cli_script.sh "show ns ip"
2 exec: show ns ip
3
4 ERROR: Invalid username or password
5
6 root@087a1e34642d:/# cli_script.sh "show ns ip" ":nsroot:
7 nsroot123"
```

8	exec: show ns ip					
9						
10	Ipaddress	Traffic		Domain	Type	Mode
	Arp	Icmp	Vserver	State		
11	-----	----	-----	-----	----	----
12	172.17.0.3	0			NetScaler IP	Active
	Enabled	Enabled	NA	Enabled		
13	192.0.0.1	0			SNIP	Active
	Enabled	Enabled	NA	Enabled		
14	Done					

- Why does SSH to NetScaler CPX fail with `root` and `nsroot` user?

From 13.0-64.35 release onwards, NetScaler CPX generates a default password and updates it for SSH users - `root` and `nsroot`. If you have not changed the password manually, password for SSH users can be found in `/var/deviceinfo/random_id` on NetScaler CPX's file-system.



© 2025 Cloud Software Group, Inc. All rights reserved. This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc. This and other products of Cloud Software Group may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>. Citrix, the Citrix logo, NetScaler, and the NetScaler logo and other marks appearing herein are either registered trademarks or trademarks of Cloud Software Group, Inc. and/or its subsidiaries in the United States and/or other countries. Other marks are the property of their respective owner(s) and are mentioned for identification purposes only. Please refer to Cloud SG's Trademark Guidelines and Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.