Hosting Applications on IOS XR

This section explains the different kinds of application hosting, and demonstrates how a simple application can be hosted natively or in a third-party container on IOS XR.

Types of Application Hosting

Application hosting on IOS XR is offered in two variants:

- **Native**—You can host applications inside the container provided by IOS XR. Applications must be built with a Cisco-specified Linux distribution (Wind River Linux 7), which uses RPM as the package manager. The applications use the libraries found in the IOS XR root file system. Configuration management tools, such as Chef and Puppet, can be used to automate the installation of the application.
- **Container**—You can create your own container on IOS XR using docker, and host applications within the container. The applications can be developed using any Linux distribution. This is well suited for applications that use system libraries that are different from that provided by the IOS XR root file system. Containers can be of two types:
    - LXC based
    - Docker based—Cisco NCS 540 supports only docker based application hosting.

Docker-Based Container Application Hosting

This section introduces the concept of container application hosting and describes its workflow.

Container application hosting makes it possible for applications to be hosted in their own environment and process space (namespace) within a Linux container on Cisco IOS XR. The application developer has complete control over the application development environment, and can use a Linux distribution of choice. The applications are isolated from the IOS XR control plane processes; yet, they can connect to networks outside XR through the XR GigE interfaces. The applications can also easily access local file systems on IOS XR.

Using Docker for Hosting Applications on Cisco IOS XR

Like an LXC, docker is a container used for hosting applications on Cisco IOS XR. Docker provides isolation for application processes from the underlying host processes on XR by using Linux network namespaces.

Need for Docker on Cisco IOS XR

Docker is becoming the industry-preferred packaging model for applications in the virtualization space. Docker provides the foundation for automating application life cycle management.

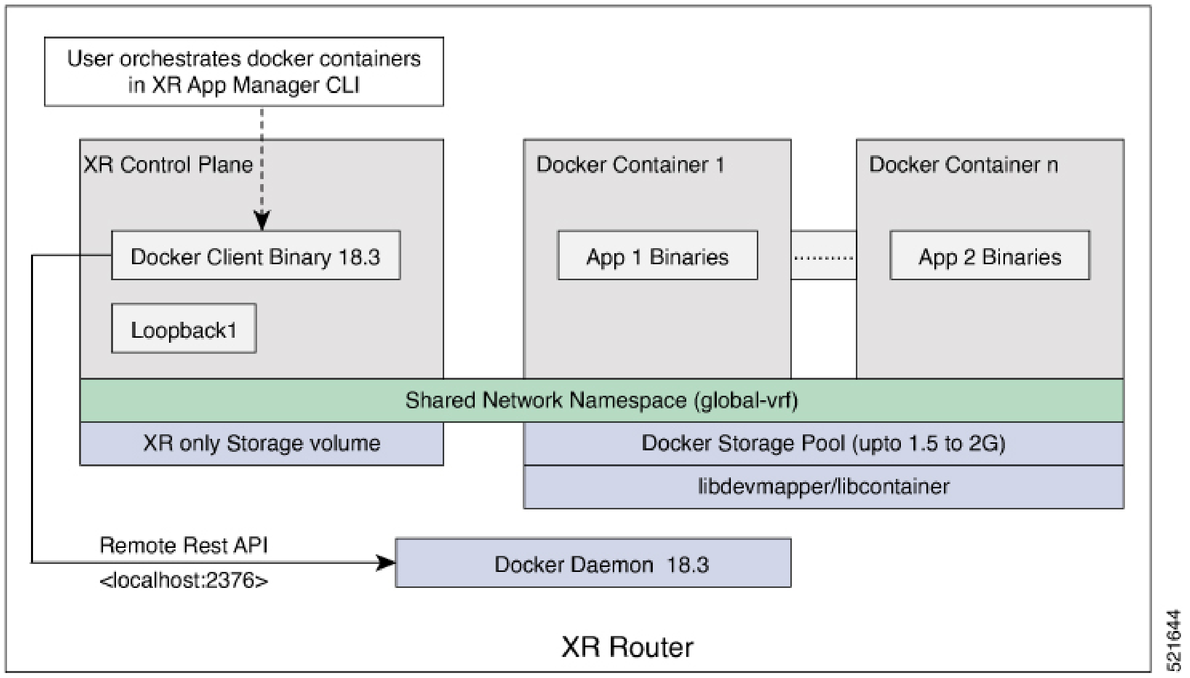
Docker follows a layered approach that consists of a base image at the bottom that supports layers of applications on top. The base images are available publicly in a repository, depending on the type of application you want to install on top. You can manipulate docker images by using the docker index and registry.

Docker provides a git-like workflow for developing container applications and supports the "thin update" mechanism, where only the difference in source code is updated, leading to faster upgrades. Docker also provides the "thin download" mechanism, where newer applications are downloaded faster because of the sharing of common base docker layers between multiple docker containers. The sharing of docker layers between multiple docker containers leads to lower footprint for docker containers on XR.

Docker Architecture on Cisco IOS XR

The following figure illustrates the docker architecture on IOS XR.

Figure 1: Docker Workflow for Updating Applications

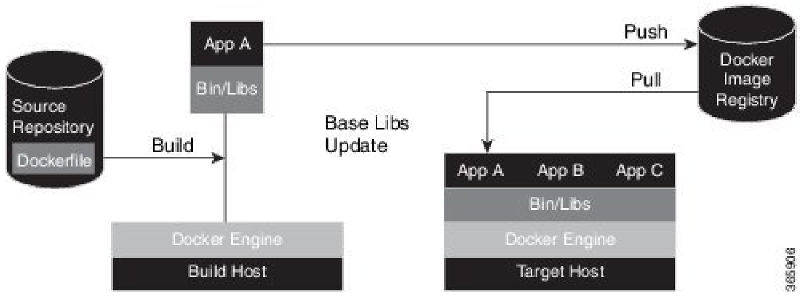

User orchestrates docker containers in XR App Manager CLI
XR Control Plane
Docker Client Binary 18.3
Loopback1
Docker Container 1
App 1 Binaries
Docker Container n
App 2 Binaries
Shared Network Namespace (global-vrf)
XR only Storage volume
Docker Storage Pool (upto 1.5 to 2G)
libdevmapper/libcontainer
Remote Rest API
<localhost:2376>
Docker Daemon 18.3
XR Router
521644

The application binaries for the applications to be hosted are installed inside the docker container.

Hosting Applications in Docker Containers

The following figure illustrates the workflow for hosting applications in Docker containers on IOS XR.

Figure 2: Docker Workflow for Application Hosting

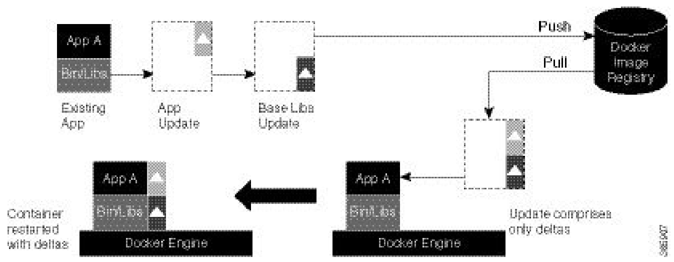


1. The docker file in the source repository is used to build the application binary file on your (docker engine build) host machine.
2. The application binary file is pushed into the docker image registry.
3. The application binary file is pulled from the docker image registry and copied to the docker container on XR (docker engine target host).
4. The application is built and hosted in the docker container on XR.

Updating Applications in Docker Containers

The following figure illustrates the workflow for updating applications hosted in docker containers.

Figure 3: Docker Workflow for Updating Applications



1. The application update is generated as a base libs update file (delta update file) and pushed to the docker image registry.
2. The delta update file (containing only the difference in application code) is pulled from the docker image registry and copied to the docker containers on XR (docker engine target host).
3. The docker containers are restarted with the delta update file.

Hosting and Seamless Activation of Third Party Applications Using Application Manager

In previous releases, the applications were hosted and controlled by the docker commands. These docker commands were executed in the bash shell of the Kernel that also hosted the Cisco IOS XR software. With the introduction of Application Manager, it is now possible to manage third-party application hosting and their functioning through Cisco IOS XR CLIs. With this feature, all the activated third party applications can restart automatically after a router reload or an RP switchover. This automatic restart of the applications ensure seamless functioning of the hosted applications.

Supported Commands on Application Manager

For every application manager command or configuration executed, the Application Manager performs the requested action by interfacing with the docker daemon through the docker socket.

The following table lists the docker container functionalities, the generic docker commands that were used in the previous releases, and its equivalent application manager commands that can now be used:

Functionality	Generic Docker Commands	Application Manager Commands
Install the application RPM	NA	`Router#appmgr package install rpm` *`image_name`*`-0.1.0-XR_7.3.1.x86_64.rpm`

Configure and activate the application	• Load image - `[xr-vm_node0_RP0_CPU0:~]$docker load -i /tmp/`*`image_name`*`.tar`
	• Verify the image on the router - `xr-vm_node0_RP0_CPU0:~]$docker images ls`
	• Create container over the image - `[xr-vm_node0_RP0_CPU0:~]$docker create` *`image_name`*
	• Start container - `[xr-vm_node0_RP0_CPU0:~]$docker start` *`my_container_id`*
		`Router#config`
		`Router(config)#appmgr`
		`Router(config-appmgr)#application` *`app_name`*
		`Router(config-application)#activate type docker source` *`image_name`* `docker-run-opts "--net=host" docker-run-cmd "iperf3 -s -d"`
		`Router(config-application)#commit`

View the list, statistics, logs, and details of the application container	• List images -`[xr-vm_node0_RP0_CPU0:~]$docker images ls`
	• List containers - `[xr-vm_node0_RP0_CPU0:~]$docker ps`
	• Statistics -`[xr-vm_node0_RP0_CPU0:~]$docker stats`
	• Logs -`[xr-vm_node0_RP0_CPU0:~]$docker logs`
		`Router#show appmgr source-table`
		`Router#show appmgr application name` *`app_name`* `info summary`
		`Router#show appmgr application name` *`app_name`* `info detail`
		`Router#show appmgr application name` *`app_name`* `stats`
		`Router#show appmgr application-table`
		`Router#show appmgr application name` *`app_name`* `logs`

Functionality	Generic Docker Commands	Application Manager Commands
Run a new command inside a running container	• Execute - `[xr-vm_node0_RP0_CPU0:~]$docker exec -it my_container_id`	`Router#appmgr application exec name app_name docker-exec-cmd`
Stop the application container	• Stop container - `[xr-vm_node0_RP0_CPU0:~]$docker stop my_container_id`	`Router#appmgr application stop name app_name`
Kill the application container	• Kill container - `[xr-vm_node0_RP0_CPU0:~]$docker kill my_container_id`	`Router#appmgr application kill name app_name`
Start the application container	• Start container - `[xr-vm_node0_RP0_CPU0:~]$docker start my_container_id`	`Router#appmgr application start name app_name`

Deactivate the application	• Stop container - `[xr-vm_node0_RP0_CPU0:~]$docker stop my_container_id`
	• Remove container - `[xr-vm_node0_RP0_CPU0:~]$docker rm my_container_id`
	• Remove image - `[xr-vm_node0_RP0_CPU0:~]$docker rmi image_name`
		`Router#configure`
		`Router(config)#no appmgr application app_name`
		`Router(config)#commit`

Uninstall the application image/RPM	• Uninstall image - `[xr-vm_node0_RP0_CPU0:~]$docker app uninstall image_name`	`Router#appmgr package uninstall package image_name-0.1.0-XR_7.3.1.x86_64`

Note The usage of the application manager commands are explained in the "Hosting iPerf in Docker Containers to Monitor Network Performance using Application Manager" section.

Configuring a Docker with Multiple VRFs

This section describes how you can configure a Docker with multiple VRFs on Cisco IOS XR. For information on configuring multiple VRFs, see Configuring Multiple VRFs for Application Hosting.

Configuration

Use the following steps to create and deploy a multi-VRF Docker on XR.

1. **Create a multi-VRF Docker with NET_ADMIN and SYS_ADMIN privileges.**

   The priviliges are required for Docker to switch namespaces and provide the Docker with all required capabilities. In the following example a Docker containing three VRFs: yellow, blue, and green is loaded on XR.

```
[XR-vm_node0_RP0_CPU0:~]$ docker run -td --net=host --name multivrfcontainer1
-v /var/run/netns/yellow:/var/run/netns/yellow
-v /var/run/netns/blue:/var/run/netns/blue
-v /var/run/netns/green:/var/run/netns/green
--cap-add NET_ADMIN --cap-add SYS_ADMIN ubuntu /bin/bash
```

Note

- Mounting the entire content of `/var/run/netns` from host to Docker is not recommended, because it mounts the content of `netns` corresponding to XR, the system admin plane, and a third-party Linux container(LXC) into the Docker.
- You should not delete a VRF from Cisco IOS XR when it is used in a Docker. If one or more VRFs are deleted from XR, the multi-VRF Docker cannot be launched.

2. **Verify if the multi-VRF Docker has been successfully loaded.**

```
[XR-vm_node0_RP0_CPU0:~]$ Docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
29c64bf812f9 ubuntu "/bin/bash" 6 seconds ago Up 4 seconds
multivrfcontainer1
```

3. **Run the multi-VRF Docker.**

```
[XR-vm_node0_RP0_CPU0:~]$ Docker exec -it multivrfcontainer1 /bin/bash
```

   By default, the Docker is loaded in global-vrf namespace on Cisco IOS XR.

4. **Verify if the multiple VRFs are accessible from the Docker.**

```
root@host:/# ifconfig
fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@host:/# ip netns list
yellow
green
blue

root@host:/# /sbin/ip netns exec green bash
root@host:/# ifconfig -a
lo        Link encap:Local Loopback
          LOOPBACK  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@host:/# ifconfig lo up
root@host:/# ifconfig lo 127.0.0.2/32
root@host:/# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.2  Mask:0.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)


[host:/misc/app_host]$ ip netns exec green bash
[host:/misc/app_host]$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.2  Mask:0.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

You have successfully launched a multi-VRF Docker on Cisco IOS XR.