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.

Application Hosting in IOS XR Container

You can create your own container on IOS XR, 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.

Selecting the Type of Application Hosting

You can select an application hosting type, depending on your requirement and the following criteria.

- **Resources**: If you need to manage the amount of resources consumed by the hosted applications, you must choose the container model, where constraints can be configured. In a native model, you can only deploy applications that use allotted resources, which are shared with internal IOS XR processes.

- **Choice of Environment**: Applications to be hosted natively must be built with the Wind River Linux 7 distribution that is offered by IOS XR. If you decide to choose the Linux distribution that is to be used for building your applications, then you must choose the container model. When you host an application using the container model, you can pre-package it prior to deployment.

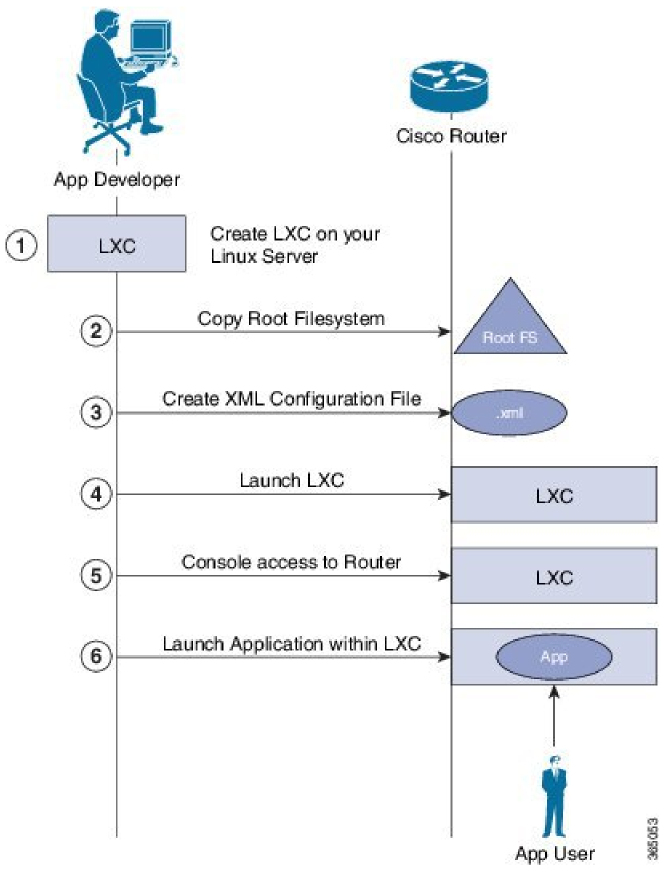
Container Application Hosting

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.

This figure illustrates the workflow for creating a Linux container for application hosting. For the complete configuration procedure, see Running iPerf as a Container Application, on page 3.

Figure 1: Container Application Hosting Workflow



There are two components in container application hosting:

- **Linux server**: This is the server you use to develop your application, to bring up the Linux Container (LXC), and to prepare the container environment.
- **Router**: This is the router running the 64-bit IOS XR that is used to host your container with the application you want to run.

1. On the Linux server, bring up the LXC, and do the following:
   a. Prepare the container environment and the required libraries.
   b. Shut down the LXC.
2. Connect to the router running IOS XR, and copy the root file system.

3. Create the configuration file for the container in .xml format. This file specifies the attributes for the container, such as name of the container, default namespace, and so on.

Note If you specify a network namespace (third-party), then by default, the LXC is launched in that namespace.

4. Launch the LXC on the router.
5. Log into the LXC on the router through IOS XR console access.
6. Manually start the application, or configure the application to start automatically when the LXC is launched.

   You can use a container, like a Linux box, to install and host applications for users.

Running iPerf as a Container Application

As an example of container application hosting, you can install an iPerf client within a LXC on IOS XR, and check its connectivity with an iPerf server installed within an LXC on another router, as described in this section.

Topology

The following illustration describes the topology used in this example.

Figure 2: iPerf as a Container Application

iPerf server is installed on Router A, and iPerf client is installed on Router B. Both installations are done within containers on the 64-bit IOS XR. The iPerf client communicates with the iPerf server through the interfaces offered by IOS XR.

Prerequisites

Ensure that you have configured the two routers as shown in the topology.

Configuration Procedure

To run iPerf as a container application, follow these steps:

1. Log into Router A, and enter the XRNNS.

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$
```

2. Launch the LXC.

```
[xr-vm_node0_RP0_CPU0:~]$virsh -c lxc+tcp://10.11.12.15:16509/ -e ^Q console demo1
```

3. Log into the LXC when prompted.

```
Connected to domain demo
Escape character is ^Q
Kernel 3.14.23-WR7.0.0.2_standard on an x86_64

host login: Password:
```

4. Install the iPerf server within the LXC on Router A.

```
[root@host ~]#apt-get install iperf
```

5. Perform Steps 1 to 4 to install the iPerf client on Router B.
6. Verify the iPerf server installation on Router A.

```
[root@host ~]#iperf -v


iperf version 2.0.5 (08 Jul 2010) pthreads
```

Similarly, verify the iPerf client installation on Router B.

7. Bind the Loopback0 interface on Router A to the iPerf server, and launch the iPerf server instance.

In this example, 1.1.1.1 is the assigned Loopback0 interface address of Router A, and 57730 is the port number used for communication.

```
[root@host ~]#iperf -s -B 1.1.1.1 -p 57730
Server listening on TCP port 57730
Binding to local address 1.1.1.1
TCP window size: 85.3 KByte (default)
```

8. Launch the iPerf client instance on Router B, by specifying the same port number used for the iPerf server, and the management IP address of Router A.

In this example, 192.168.122.213 is the management IP address of Router A, and 57730 is the port number used to access the iPerf server.

```
[root@host ~]#iperf -c 192.168.122.213 -p 57730
------------------------------------------------------------
Client connecting to 192.168.122.213, TCP port 57730
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 46974 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   146 MBytes   122 Mbits/sec
```

To use UDP, instead of TCP, to communicate with the iPerf server, use the following command.

```
[root@host ~]#iperf -c 192.168.122.213 -p 57730 -u
------------------------------------------------------------
Client connecting to 192.168.122.213, UDP port 57730
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 41466 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.233 ms    0/  893 (0%)
[root@hostB ~]#
```

9. Ping the iPerf server from the iPerf client on Router B.

```
[root@host ~]#/bin/ping 192.164.168.10
PING 192.164.168.10 (192.164.168.10) 56(84) bytes of data.
```

```
64 bytes from 192.164.168.10: icmp_seq=1 ttl=255 time=13.0 ms
64 bytes from 192.164.168.10: icmp_seq=2 ttl=255 time=2.14 ms
64 bytes from 192.164.168.10: icmp_seq=3 ttl=255 time=2.21 ms
```

The iPerf client hosted on Router B can access the iPerf server hosted on Router A.

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.

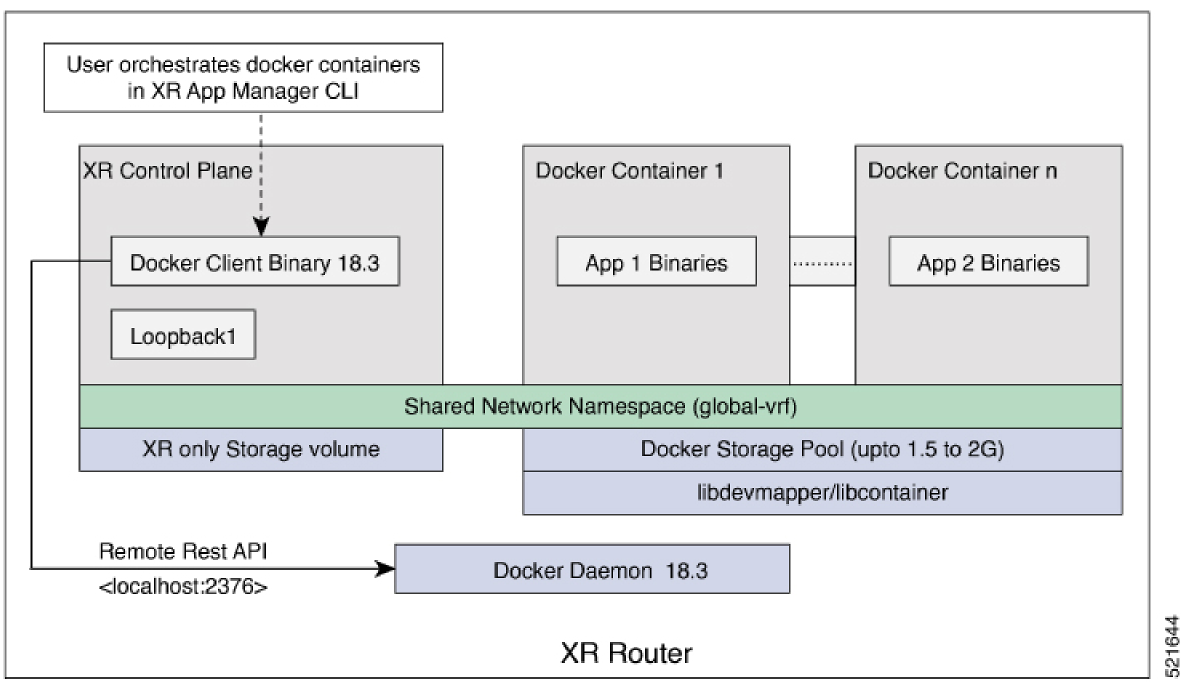
Docker follows a layered approach that consists of a base image at the bottom that supports layers of applications on top. The base images are available publicly in a repository, depending on the type of application you want to install on top. You can manipulate docker images by using the docker index 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 3: Docker Workflow for Updating Applications

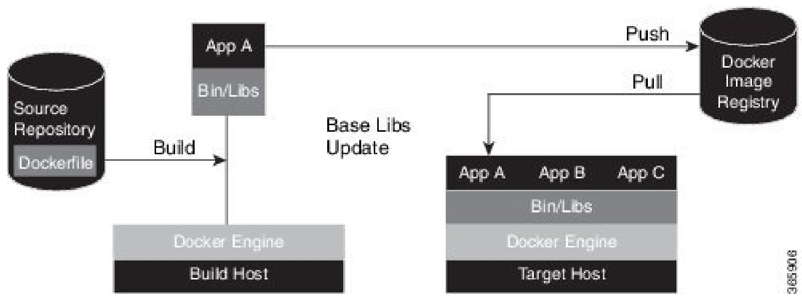


The application binaries for the applications to be hosted are installed inside the docker container.

Hosting Applications in Docker Containers

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

Figure 4: Docker Workflow for Application Hosting

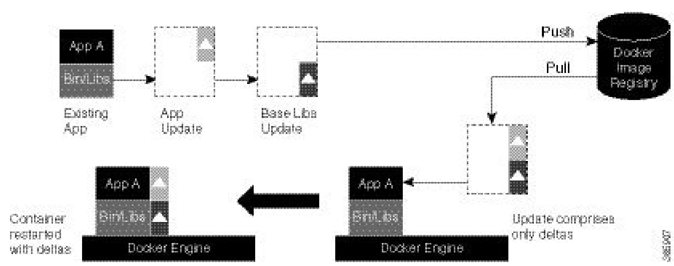


1. The docker file in the source repository is used to build the application binary file on your (docker engine build) host machine.
2. The application binary file is pushed into the docker image registry.
3. The application binary file is pulled from the docker image registry and copied to the docker 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 5: 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.

TPA security

IOS XR is equipped with inherent safeguards to prevent third party applications from interfering with its role as a Network OS.

Although IOS XR doesn't impose a limit on the number of TPAs that can run concurrently, it does impose constraints on the resources allocated to the Docker daemon, based on the following parameters:

- CPU: By default, ¼ of the CPU per core available in the platform.

  Starting from IOS XR Release 24.4.1, you can hard limit the default CPU usage in the range between 25-75% of the total system CPU using the **appmgr resources containers limit cpu** *value*value command. This configuration restricts the TPAs from using more CPU than the set hard limit value irrespective of the CPU usage by other XR processes.

  This example provides the CPU hard limit configuraion.

```
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit cpu ?
  <25-75>  In Percentage
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit cpu 25
```

- RAM: By default, 1 GB of memory is available.

  Starting from IOS XR Release 24.4.1, you can hard limit the default memory usage in the range between 1-25% of the overall system memory using the **appmgr resources containers limit memory** *value* command. This configuration restricts the TPAs from using more memory than the set hard limit value.

  This example provides the memory hard limit configuration.

```
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit memory ?
  <1-25>  In Percentage
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit memory 20
```

Hosting and Seamless Activation of Third Party Applications Using Application Manager

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Hosting and Seamless Activation of Third Party Applications Using Application Manager	Release 7.3.2	Application Manager manages third-party application hosting and their functioning through Cisco IOS XR CLIs. With this feature, all the activated third party applications are automatically restarted after a router reload or an RP switchover. This process ensures seamless functioning of the hosted applications. Prior to this release, the hosted applications were controlled by the Docker commands executed in the bash shell of the Kernel that hosts the Cisco IOS XR software.

Feature Name	Release Information	Feature Description
On-Demand Docker Daemon Service for Hosting Applications	Release 7.5.1	From this release onwards, the Docker daemon service starts on a router only if you configure a third-party hosting application using the **appmgr** command. Such an on-demand service optimizes operating system resources such as CPU, memory, and power. In earlier releases, the Docker daemon service automatically started during the router boot up.

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`**

Functionality	Generic Docker Commands	Application Manager Commands

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`

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.

Customize Docker Run Options Using Application Manager

Table 2: Feature History Table

Feature Name	Release Information	Description
Customize Docker Run Options Using Application Manager	Release 24.1.1	Introduced in this release on: NCS 5500 fixed port routers; NCS 5700 fixed port routers; NCS 5500 modular routers (NCS 5500 line cards; NCS 5700 line cards [Mode: Compatibility; Native]) You can now leverage Application Manager to efficiently overwrite default docker runtime configurations, tailoring them to specific parameters like CPU usage, security settings, and health checks. You can thus optimize application performance, maintain fair resource allocation among multiple dockers, and establish non-default network security settings to meet specific security requirements. Additionally, you can accurately monitor and reflect the health of individual applications. This feature modifies the **docker-run-opts** option command.

With this feature, runtime options for docker containerized applications on IOS-XR can be configured during launch using the **appmgr activate**" command. AppMgr, which oversees docker containerized applications, ensures that these runtime options can effectively override default configurations, covering aspects like CPU, security, and health checks during the container launch.

This feature introduces multiple runtime options that allow users to customize different parameters of docker containers. The configuration of these runtime options is flexible, as users can use either command or Netconf for the configuration process. Regardless of the chosen method, runtime options must be added to **docker-run-opts** as needed.

The following are the docker run option commands introduced in IOS-XR software release 24.1.1.

Table 3: Docker Run Options

Docker Run Option	Description
--cpus	Number of CPUs
--cpuset-cpus	CPUs in which to allow execution (0-3, 0,1)

Docker Run Option	Description
--cap-drop	Drop Linux capabilities
--user, -u	Sets the username or UID
--group-add	Add additional groups to run
--health-cmd	Run to check health
--health-interval	Time between running the check
--health-retries	Consecutive failures needed to report unhealthy
--health-start-period	Start period for the container to initialize before starting health-retries countdown
--health-timeout	Maximum time to allow one check to run
--no-healthcheck	Disable any container-specified HEALTHCHECK
--add-host	Add a custom host-to-IP mapping (host:ip)
--dns	Set custom DNS servers
--dns-opt	Set DNS options
--dns-search	Set custom DNS search domains
--domainname	Container NIS domain name
--oom-score-adj	Tune host's OOM preferences (-1000 to 1000)
--shm-size	Option to set the size of /dev/shm
--init	Run an init inside the container that forwards signals and reaps processes
--label, -l	Set meta data on a container
--label-file	Read in a line delimited file of labels
--pids-limit	Tune container pids limit (set -1 for unlimited)
--work-dir	Working directory inside the container
--ulimit	Ulimit options
--read-only	Mount the container's root filesystem as read only
--volumes-from	Mount volumes from the specified container(s)
--stop-signal	Signal to stop the container
--stop-timeout	Timeout (in seconds) to stop a container
--cap-addNET_RAW	Enable NET_RAW capabilities

Prior to IOS-XR software release 24.1.1, only the below mentioned docker run option commands were supported.

Table 4: Docker Run Options

Docker Run Option	Description
--publish	Publish a container's port(s) to the host
--entrypoint	Overwrite the default ENTRYPOINT of the image
--expose	Expose a port or a range of ports
--link	Add link to another container
--env	Set environment variables
--env-file	Read in a file of environment variables
--network	Connect a container to a network
--hostname	Container host name
--interactive	Keep STDIN open even if not attached
--tty	Allocate a pseudo-TTY
--publish-all	Publish all exposed ports to random ports
--volume	Bind mount a volume
--mount	Attach a filesystem mount to the container
--restart	Restart policy to apply when a container exits
--cap-add	Add Linux capabilities
--log-driver	Logging driver for the container
--log-opt	Log driver options
--detach	Run container in background and print container ID
--memory	Memory limit
--memory-reservation	Memory soft limit
--cpu-shares	CPU shares (relative weight)
--sysctl	Sysctl options

Guidelines and Limitations

- For the options `--mount` and `--volume`, only the following values can be configured:
    - "/var/run/netns"

- "/var/lib/docker"
- "/misc/disk1"
- "/disk0"

For eXR platforms:

- "/var/run/netns"
- "/misc/app_host"
- "/misc/disk1"
- "/disk0"

- The maximum allowed size for shm-size option is 64 Mb.
- Prior to Release 24.4.1, all container logs were recorded with an info severity level (sev-6), regardless of the docker run time options used. From Release 24.4.1, if the docker run time option `-it` is used, the container logs are generated with an info severity level (sev-6). However, if the `--it` option is not included, the logs are produced with an error severity level (sev-3).
- From Release 24.4.1, you can use the rsyslog daemon to forward syslog messages to remote syslog servers. To know more, see Support for logging functionality on third-party applications.

Configuration

This section provides the information on how to configure the docker run time options.

In this example we configure the docker run time option **--pids-limit** to limit the number of process IDs using appmgr.

```
Router#appmgr application alpine_app activate type docker source alpine docker-run-opts
"-it -pids-limit 90" docker-run-cmd "sh"
Router#
```

In this example we configure the docker run time option **--pids-limit** to limit the number of process IDs using Netconf.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <appmgr xmlns=http://cisco.com/ns/yang/Cisco-IOS-XR-um-appmgr-cfg>

        <applications>
          <application>
            <application-name>alpine_app</application-name>
            <activate>
              <type>docker</type>
                                             <source-name>alpine</source-name>
                                             <docker-run-cmd>/bin/sh</docker-run-cmd>
                                             <docker-run-opts>-it
--pids-limit=90</docker-run-opts>
            </activate>
          </application>
        </applications>
      </appmgr>
```

```
    </config>
  </edit-config>
```

Verification

This example shows how to verify the docker run time option configuration.

```
Router# show running-config appmgr
Thu Mar 23 08:22:47.014 UTC
appmgr
 application alpine_app
  activate type docker source alpine docker-run-opts "-it –pids-limit 90" docker-run-cmd
"sh"
 !
!
```

You can also use **docker inspect** *container id* to verify the docker run time option configuration.

```
Router# docker inspect 25f3c30eb424
[
    {
                  "PidsLimit": 90,
    }
]
```

Docker Application Management using IPv6 Address

Table 5: Feature History Table

Feature Name	Release Information	Description
Docker Application Management using IPv6 Address	Release 7.11.1	Introduced in this release on: NCS 5500 fixed port routers; NCS 5700 fixed port routers; NCS 5500 modular routers (NCS 5500 line cards; NCS 5700 line cards [Mode: Compatibility; Native]) In this release, you gain the ability to manage Docker applications within containers using IPv6 addresses via the router's management interface. Leveraging IPv6 addresses provides expanded addressing options, enhances network scalability, and enables better segmentation and isolation of applications within the network. Prior to this update, only IPv4 addresses could be used to manage docker applications.

The Application Manager in IOS-XR software release 7.3.15 introduces support for an application networking feature that facilitates traffic forwarding across Virtual Routing and Forwarding (VRF) instances. This feature is implemented through the deployment of a relay agent contained within an independent docker container.

The relay agent acts as a bridge, connecting two network namespaces within the host system and actively transferring traffic between them. Configurations can be made to establish forwarding between either a single pair of ports or multiple pairs, based on your network requirements.

One of the main uses of this feature is to allow the management of Linux-based Docker applications that are running in the default VRF through a management interface. This management interface can be located in a separate VRF. This feature ensures that Docker applications can be managed seamlessly across different VRFs.

In the IOS-XR software release 7.11.1, enhanced management capabilities are offered for docker applications. Now, you can leverage IPv6 addresses to manage applications within docker containers via the management interface of the Cisco 8000 router. This update provides improved accessibility and control over your Docker applications using IPv6 addressing. Prior to the IOS-XR software release 7.11.1, application management for docker containers could only be conducted using IPv4 addresses.

Restrictions and Limitations

In configuring your setup, please consider the following restrictions and limitations:

- **VRF Forwarding Limitation**: The Virtual Routing and Forwarding (VRF) is only supported for Docker apps with host networking.
- **Relay Agent Availability and Management**: The relay agent container is designed to be highly available. It will be managed by the Application Manager (App Mgr).
- **Relay Agent Creation**: For each pair of forwarded ports, one relay agent container will be created.
- **Port Limitation per Application**: The total effective number of ports for each application is limited to a maximum of 10.

Configure VRF Forwarding

To manage a Docker application using the Application Manager through the Management Interface, follow these steps:

Procedure

Step 1 **Configure the app manager**: The application manager is configured to access the docker application. Use the **appmgr application***application-name* keyword to enable and specify configuration parameters for the VRF forwarding. A typical example would look like this:

Example:

```
Router#appmgr
Router#application Testapp
```

Note
The VRF forwarding related run options like **--vrf-forward** and **--vrf-forward-ip-range** will not be passed to the Docker engine when the app container is run.

Step 2 **Enable Basic Forwarding Between Two Ports**: To enable traffic forwarding between two ports in different VRFs, use the following configuration:

Example:

```
Router#activate type docker source swanagent docker-run-opts "--vrf-forward vrf-mgmt:5001
vrf-default:8001 --net=host -it"
```

This command enables traffic on port 5000 at all addresses in vrf-mgmt to be forwarded to the destination veth device in vrf-default on port 8000.

To enable VRF forwarding between multiple ports, follow the steps below:

- **Enable Forwarding Between a Range of Ports**: To enable traffic forwarding between port ranges in different VRFs, use the following configuration:

  ```
  Router#--vrf-forward vrf-mgmt:5000-5002 vrf-default:8000-8002
  ```

  This command enables traffic on ports 5000, 5001, and 5002 at all addresses in vrf-mgmt to be forwarded to the destination veth device in vrf-default on ports 8000, 8001, and 8002 respectively.

- **Enable Forwarding Between Multiple VRF Pairs or Port Ranges**: To enable traffic forwarding between multiple VRF pairs, use multiple **--vrf-forward** command.

  ```
  Router#--vrf-forward vrf-mgmt:5000 vrf-default:8000 --vrf-forward vrf-mgmt:5003-5004
  vrf-default:8003-8004
  Router#--vrf-forward vrf-mgmt1:5000 vrf-default:8000 --vrf-forward vrf-mgmt2:5000 vrf-default:8001
  ```

  You can provide any number of --vrf-forward options, but the total number of port pairs involved should not exceed 10.

Verifying VRF Forwarding for Application Manager

To verify the VRF forwarding, follow these steps:

SUMMARY STEPS

1. **Check the running configuration of the app manager**: Use the **show running-config appmgr** keyword to verify the VRF forwarding. A typical example would look like this:

DETAILED STEPS

Procedure

	Command or Action	Purpose

Step 1	**Check the running configuration of the app manager**: Use the **show running-config appmgr** keyword to verify the VRF forwarding. A typical example would look like this:	`Router#show running-config appmgr`
		`Thu Oct 26 12:04:06.063 UTC`
		`appmgr`
		` application swan`
		` activate type docker source swanagent docker-run-opts "--vrf-forward vrf-management:11111 vrf-default:10000 -it --restart always --cap-add=SYS_ADMIN --net=host --log-opt max-size=20m --log-opt max-file=3 -e HOSTNAME=$HOSTNAME -v /var/run/netns:/var/run/netns -v {app_install_root}/config/swanagent:/root/config -v {app_install_root}/config/swanagent/hostname:/etc/hostname`

	Command or Action	Purpose

		` -v /var/lib/docker/ems/grpc.sock:/root/grpc.sock"`
		
		` !`
		`!`
		
		In this example, port 11111 is assigned as the management port and port 10000 is the VRF port in the container.

Using Vagrant for Hosting Applications

You can use vagrant on a host device of your choice, for hosting applications as described in the following sections.

Note IOS-XR software version 6.x.x and above is not supported on Vagrant.

Pre-requisites for Using Vagrant

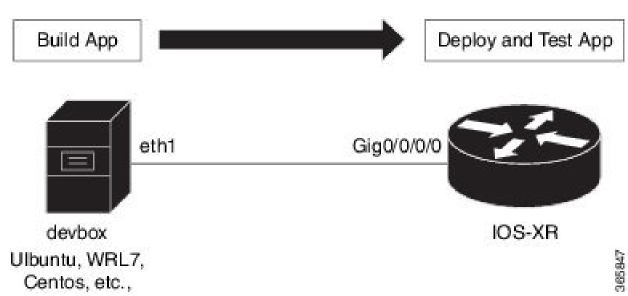
Before you can start using vagrant, ensure that you have fulfilled the following requirements on your host device.

- Latest version of Vagrant for your operating system. We recommend Version 1.8.6.
- Latest version of a virtual box for your operating system. We recommend Version 5.1+.
- Minimum of 5 GB of RAM with two cores.
- (Optional) If you are using the Windows Operating System, we recommend that you download the Git bash utility for running the commands.

Setting up an Application Development Topology By Using Vagrant

For the sake of illustration, we will use a simple two-node topology, where an instance of Cisco IOS XR behaves as one node (`rtr`), and an instance of Ubuntu (hypervisor) behaves as the other (`devbox`). We will use the `devbox` to develop the app topology and deploy it on the `rtr`.

Figure 6: Application Development Topology



Procedure

To create an application development topology on vagrant, follow these steps.

1. Generate an API key and a CCO ID by using the steps described on Github.
2. Download the latest stable version of the IOS-XRv vagrant box.

```
$ curl <cco-id>:<API-KEY>

$ BOXURL --output ~/iosxrv-fullk9-x64.box

$ vagrant box add --name IOS-XRv ~/iosxrv-fullk9-x64.box
```

3. Verify if the vagrant box has been successfully installed.

```
annseque@ANNSEQUE-WS02 MINGW64 ~ vagrant box list
IOS-XRv (virtualbox, 0)
```

4. Create a working directory.

```
annseque@ANNSEQUE-WS02 MINGW64 ~ mkdir ~/iosxrv
annseque@ANNSEQUE-WS02 MINGW64 ~ cd ~/iosxrv
```

5. Initialize the vagrant file with the new vagrant box.

```
ANNSEQUE-WS02 MINGW64:iosxrv annseque$ vagrant init IOS-XRv
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

6. Clone the `vagrant-xrdocs` repository.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ git clone https://github.com/ios-xr/vagrant-xrdocs.git
```

7. Navigate to the `vagrant-xrdocs` repository and locate the `lxc-app-topo-bootstrap` directory.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ cd vagrant-xrdocs/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ ls
ansible-tutorials/       native-app-topo-bootstrap/  simple-mixed-topo/
lxc-app-topo-bootstrap/  README.md                   single_node_bootstrap/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ ls lxc-app-topo-bootstrap/
configs/  scripts/  Vagrantfile
```

8. (Optional) View the contents of the vagrant file in the `lxc-app-topo-bootstrap` directory.

   The vagrant file (`Vagrantfile`) contains the two node topology for application development. You can modify this by using a vi editor, if required.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd lxc-app-topo-bootstrap/
```

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ cat Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

Vagrant.configure(2) do |config|


   config.vm.define "rtr" do |node|
      node.vm.box =  "IOS-XRv"

      # gig0/0/0 connected to "link1"
      # auto_config is not supported for XR, set to false

      node.vm.network :private_network, virtualbox__intnet: "link1", auto_config
                                                                                 :
false


      #Source a config file and apply it to XR

      node.vm.provision "file", source: "configs/rtr_config", destination: "/hom

e/vagrant/rtr_config"

      node.vm.provision "shell" do |s|
          s.path =  "scripts/apply_config.sh"
          s.args = ["/home/vagrant/rtr_config"]
      end

    end


    config.vm.define "devbox" do |node|
      node.vm.box =  "ubuntu/trusty64"

      # eth1 connected to link1
      # auto_config is supported for an ubuntu instance

      node.vm.network :private_network, virtualbox__intnet: "link1", ip: "11.1.1

.20"

    end

end
```

You have successfully created an application development topology on vagrant. See Deploying an Application Development Topology by Using Vagrant, on page 21 for information on deploying the topology on vagrant.

Deploying an Application Development Topology by Using Vagrant

This section describes how you can deploy an application development topology on vagrant for creating and hosting your applications.

Procedure

To deploy an application development topology on vagrant, follow these steps.

Note Ensure you have created an application development topology as described in Setting up an Application Development Topology By Using Vagrant, on page 19, before proceeding with the following steps.

1. Ensure you are in the `lxc-app-topo-bootstrap` directory, and launch the vagrant instance.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant up

Bringing machine 'rtr' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
==> rtr: Clearing any previously set forwarded ports...
==> rtr: Clearing any previously set network interfaces...
==> rtr: Preparing network interfaces based on configuration...
    rtr: Adapter 1: nat
    rtr: Adapter 2: intnet
==> rtr: Forwarding ports...
    rtr: 57722 (guest) => 2222 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
==> rtr: Running 'pre-boot' VM customizations...
==> rtr: Booting VM...
==> rtr: Waiting for machine to boot. This may take a few minutes...
    rtr: SSH address: 127.0.0.1:2222
    rtr: SSH username: vagrant
    rtr: SSH auth method: private key
    rtr: Warning: Remote connection disconnect. Retrying...
    ...
==> rtr: Machine booted and ready!
==> rtr: Checking for guest additions in VM...
    rtr: No guest additions were detected on the base box for this VM! Guest
    rtr: additions are required for forwarded ports, shared folders, host only
    rtr: networking, and more. If SSH fails on this machine, please install
    rtr: the guest additions and repackage the box to continue.
    rtr:
    rtr: This is not an error message; everything may continue to work properly,
    rtr: in which case you may ignore this message.
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: flag to force provisioning. Provisioners marked to run always will still run.
==> devbox: Checking if box 'ubuntu/trusty64' is up to date...
==> devbox: A newer version of the box 'ubuntu/trusty64' is available! You currently
==> devbox: have version '20160801.0.0'. The latest is version '20160826.0.1'. Run
==> devbox: `vagrant box update` to update.
==> devbox: Clearing any previously set forwarded ports...
==> devbox: Fixed port collision for 22 => 2222. Now on port 2200.
==> devbox: Clearing any previously set network interfaces...
==> devbox: Preparing network interfaces based on configuration...
    devbox: Adapter 1: nat
    devbox: Adapter 2: intnet
==> devbox: Forwarding ports...
    devbox: 22 (guest) => 2200 (host) (adapter 1)
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2200
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    devbox: Warning: Remote connection disconnect. Retrying...
    devbox: Warning: Remote connection disconnect. Retrying...
```

```
==> devbox: Machine booted and ready!
==> devbox: Checking for guest additions in VM...
    devbox: The guest additions on this VM do not match the installed version of
    devbox: VirtualBox! In most cases this is fine, but in rare cases it can
    devbox: prevent things such as shared folders from working properly. If you see
    devbox: shared folder errors, please make sure the guest additions within the
    devbox: virtual machine match the version of VirtualBox you have installed on
    devbox: your host and reload your VM.
    devbox:
    devbox: Guest Additions Version: 4.3.36
    devbox: VirtualBox Version: 5.0
==> devbox: Configuring and enabling network interfaces...
==> devbox: Mounting shared folders...
    devbox: /vagrant => C:/Users/annseque/vagrant-xrdocs/lxc-app-topo-bootstrap
==> devbox: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> devbox: flag to force provisioning. Provisioners marked to run always will still
run.

==> rtr: Machine 'rtr' has a post `vagrant up` message. This is a message
==> rtr: from the creator of the Vagrantfile, and not from Vagrant itself:
==> rtr:
==> rtr:
==> rtr:     Welcome to the IOS XRv (64-bit) Virtualbox.
==> rtr:     To connect to the XR Linux shell, use: 'vagrant ssh'.
==> rtr:     To ssh to the XR Console, use: 'vagrant port' (vagrant version > 1.8)
==> rtr:     to determine the port that maps to guestport 22,
==> rtr:     then: 'ssh vagrant@localhost -p <forwarded port>'
==> rtr:
==> rtr:     IMPORTANT:  READ CAREFULLY
==> rtr:     The Software is subject to and governed by the terms and conditions
==> rtr:     of the End User License Agreement and the Supplemental End User
==> rtr:     License Agreement accompanying the product, made available at the
==> rtr:     time of your order, or posted on the Cisco website at
==> rtr:     www.cisco.com/go/terms (collectively, the 'Agreement').
==> rtr:     As set forth more fully in the Agreement, use of the Software is
==> rtr:     strictly limited to internal use in a non-production environment
==> rtr:     solely for demonstration and evaluation purposes. Downloading,
==> rtr:     installing, or using the Software constitutes acceptance of the
==> rtr:     Agreement, and you are binding yourself and the business entity
==> rtr:     that you represent to the Agreement. If you do not agree to all
==> rtr:     of the terms of the Agreement, then Cisco is unwilling to license
==> rtr:     the Software to you and (a) you may not download, install or use the
==> rtr:     Software, and (b) you may return the Software as more fully set forth
==> rtr:     in the Agreement.
```

You have successfully deployed the two nodes, `rtr` and `devbox` on your host machine.

2. To access the XR router console, check the port number that maps to the guest port number 22.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant port rtr
The forwarded ports for the machine are listed below. Please note that
these values may differ from values configured in the Vagrantfile if the
provider supports automatic port collision detection and resolution.

    22 (guest) => 2223 (host)
 57722 (guest) => 2222 (host)
```

You need to use port number 2223 to SSH to the `rtr` node (XR).

3. Access the XR router console (`rtr` console) through SSH.

The password for `vagrant@localhost` is **vagrant**.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:


RP/0/RP0/CPU0:ios#
```

You are at the XR router console, or the console of the `rtr` node in this example.

4. Check the GigE interface IP address of the `rtr`.

   You will need the GigE interface IP address to access the `rtr` console from the `devbox` console at a later stage.

```
RP/0/RP0/CPU0:ios# show ipv4 interface gigabitEthernet 0/0/0/0 brief
Wed Aug 31 04:00:48.006 UTC

Interface                      IP-Address      Status                Protocol
GigabitEthernet0/0/0/0         11.1.1.10       Up                    Up
```

Note To access the XR Linux shell from the `rtr` console, use the **run** command.

```
RP/0/RP0/CPU0:ios# run
Wed Aug 31 04:01:45.119 UTC

[xr-vm_node0_RP0_CPU0:~]$
```

5. Exit the `rtr` console, and access the `devbox` console through SSH.

```
RP/0/RP0/CPU0:ios# exit
Connection to localhost closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$  vagrant ssh devbox

Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

 System information disabled due to load higher than 1.0

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

25 packages can be updated.
12 updates are security updates.


vagrant@vagrant-ubuntu-trusty-64:~$
```

6. Verify if you can access the `rtr` console from the `devbox` console, by pinging the GigE interface of the `rtr`.

   Use the GigE interface IP address you retrieved in Step 12.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ping 11.1.1.10 -c 2
PING 11.1.1.10 (11.1.1.10) 56(84) bytes of data.
```

```
64 bytes from 11.1.1.10: icmp_seq=1 ttl=255 time=40.2 ms
64 bytes from 11.1.1.10: icmp_seq=2 ttl=255 time=6.67 ms

--- 11.1.1.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 6.670/23.457/40.245/16.788 ms
vagrant@vagrant-ubuntu-trusty-64:~$
```

Note To access the XR Linux console, exit the `devbox` console and run the **vagrant ssh rtr** command from the `lxc-app-topo-bootstrap` directory.

```
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh rtr
Last login: Thu Jul 21 05:51:28 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$
```

You have successfully deployed an application development topology on your host device.

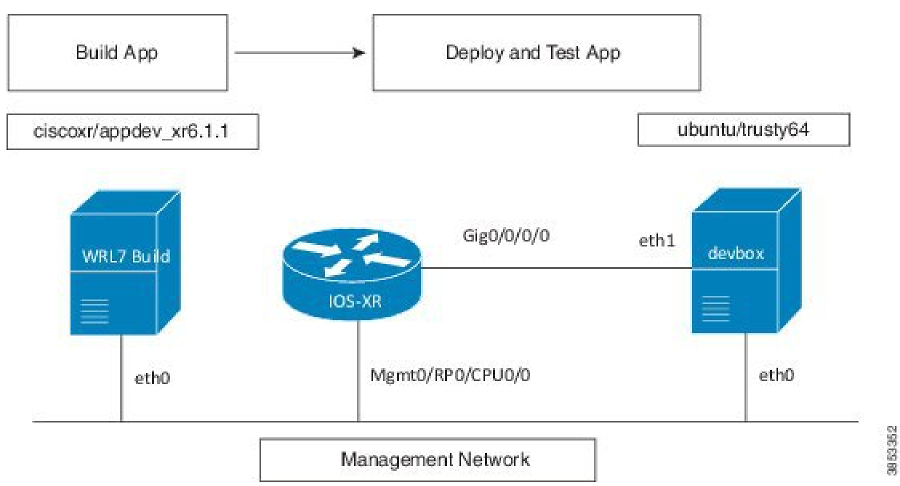
Hosting a Wind River Linux (WRL7) Application Natively By Using Vagrant

This section describes how you can host a Wind river Linux (WRL7) application natively by using vagrant.

Native Application Hosting Topology

For the sake of illustration, we will use the three vagrant instance topology as shown in the following figure.

Figure 7: Native Application Hosting Topology on a Vagrant Box



Procedure

Use the following steps to host an application natively on IOS XR.

Note Ensure you have created an application development topology as described in Setting up an Application Development Topology By Using Vagrant, on page 19, before proceeding with the following steps.

1. Verify if you have the `IOS-XRv` and the `ciscoxr/appdev-xr6.1.1` vagrant boxes installed on your machine.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ vagrant box list
IOS-XRv                  (virtualbox, 0)
ciscoxr/appdev-xr6.1.1 (virtualbox, 1.0)
ubuntu/trusty64          (virtualbox, 20160602.0.0)
```

2. Clone the `vagrant-xrdocs` repository.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ git clone https://github.com/ios-xr/vagrant-xrdocs.git
```

3. Navigate to the `vagrant-xrdocs/native-app-topo-bootstrap` directory and launch the vagrant instance.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd native-app-topo-bootstrap/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant up

Bringing machine 'rtr' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'wrl7_build' up with 'virtualbox' provider...
==> rtr: Clearing any previously set forwarded ports...
==> rtr: Clearing any previously set network interfaces...
==> rtr: Preparing network interfaces based on configuration...
    rtr: Adapter 1: nat
    rtr: Adapter 2: intnet
==> rtr: Forwarding ports...
    rtr: 57722 (guest) => 2222 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
==> rtr: Running 'pre-boot' VM customizations...
==> rtr: Booting VM...
==> rtr: Waiting for machine to boot. This may take a few minutes...
    rtr: SSH address: 127.0.0.1:2222
    rtr: SSH username: vagrant
    rtr: SSH auth method: private key
    rtr: Warning: Remote connection disconnect. Retrying...
 ...
==> rtr: Machine booted and ready!
==> rtr: Checking for guest additions in VM...
    rtr: No guest additions were detected on the base box for this VM! Guest
    rtr: additions are required for forwarded ports, shared folders, host only
    rtr: networking, and more. If SSH fails on this machine, please install
    rtr: the guest additions and repackage the box to continue.
    rtr:
    rtr: This is not an error message; everything may continue to work properly,
    rtr: in which case you may ignore this message.
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: flag to force provisioning. Provisioners marked to run always will still run.
==> devbox: Checking if box 'ubuntu/trusty64' is up to date...
==> devbox: A newer version of the box 'ubuntu/trusty64' is available! You currently
==> devbox: have version '20160801.0.0'. The latest is version '20160907.0.0'. Run
==> devbox: `vagrant box update` to update.
==> devbox: Clearing any previously set forwarded ports...
```

```
==> devbox: Fixed port collision for 22 => 2222. Now on port 2200.
==> devbox: Clearing any previously set network interfaces...
==> devbox: Preparing network interfaces based on configuration...
    devbox: Adapter 1: nat
    devbox: Adapter 2: intnet
==> devbox: Forwarding ports...
    devbox: 22 (guest) => 2200 (host) (adapter 1)
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2200
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    devbox: Warning: Remote connection disconnect. Retrying...
    devbox: Warning: Remote connection disconnect. Retrying...
==> devbox: Machine booted and ready!
...
==> wrl7_build: Checking if box 'ciscoxr/appdev-xr6.1.1' is up to date...
==> wrl7_build: Clearing any previously set forwarded ports...
==> wrl7_build: Fixed port collision for 22 => 2222. Now on port 2201.
==> wrl7_build: Clearing any previously set network interfaces...
==> wrl7_build: Preparing network interfaces based on configuration...
    wrl7_build: Adapter 1: nat
==> wrl7_build: Forwarding ports...
    wrl7_build: 22 (guest) => 2201 (host) (adapter 1)
==> wrl7_build: Booting VM...
==> wrl7_build: Waiting for machine to boot. This may take a few minutes...
    wrl7_build: SSH address: 127.0.0.1:2201
    wrl7_build: SSH username: vagrant
    wrl7_build: SSH auth method: private key
    wrl7_build: Warning: Remote connection disconnect. Retrying...
...
==> wrl7_build: Welcome to the IOS XR Application Development (AppDev) VM that provides
 a WRL7 based native environment to build appli
                                                        cations for IOS XR (Release
 6.1.1) platforms.
```

4. Verify if the WRL7 build instance has launched.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant status
Current machine states:

rtr                        running (virtualbox)
devbox                     running (virtualbox)
wrl7_build                 running (virtualbox)
...
```

5. Access the WRL7 build instance through SSH, and retrieve the source code of the application you want to host natively.

   In this example, we fetch the source code for the iPerf application.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant ssh wrl7_build

-------------------------------------------------------------------------------------

localhost:~$ wget https://iperf.fr/download/source/iperf-2.0.9-source.tar.gz
--2016-09-13 01:54:58--  https://iperf.fr/download/source/iperf-2.0.9-source.tar.gz
Resolving iperf.fr... 194.158.119.186, 2001:860:f70a::2
Connecting to iperf.fr|194.158.119.186|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 277702 (271K) [application/x-gzip]
Saving to: 'iperf-2.0.9-source.tar.gz'
```

```
100%[===================================================================>] 277,702
 153KB/s   in 1.8s

2016-09-13 01:55:01 (153 KB/s) - 'iperf-2.0.9-source.tar.gz' saved [277702/277702]

-------------------------------------------------------------------------------------

localhost:~$ ls
iperf-2.0.9-source.tar.gz
localhost:~$
```

6. Copy the source code tar ball to the `/usr/src/rpm/SOURCES/` build location.

```
localhost:~$ sudo cp /home/vagrant/iperf-2.0.9-source.tar.gz /usr/src/rpm/SOURCES/
```

7. Retrieve the XML spec file (`iperf.spec`) for building the RPM.

```
localhost:~$ wget http://10.30.110.214/iperf.spec
--2016-09-13 01:58:44--  http://10.30.110.214/iperf.spec
Connecting to 10.30.110.214:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 609
Saving to: 'iperf.spec'

100%[=============================================================>] 609        --.-K/s
   in 0s

2016-09-13 01:58:45 (38.2 MB/s) - 'iperf.spec' saved [609/609]

-------------------------------------------------------------------------------------
localhost:~$ ls
iperf-2.0.9-source.tar.gz  iperf.spec
```

8. Build the RPM by using the retrieved spec file.

```
localhost:~$ sudo rpmbuild -ba iperf.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.59743
+ umask 022
+ cd /usr/lib64/rpm/../../src/rpm/BUILD
+ cd /usr/src/rpm/BUILD
+ rm -rf iperf-2.0.9
+ /bin/tar -xf -
...
Requires: libc.so.6()(64bit) libc.so.6(GLIBC_2.14)(64bit) libc.so.6(GLIBC_2.2.5)(64bit)

 libc.so.6(GLIBC_2.3)(64bit) libc.so.6(GLIBC_2.7)(64bit)
 libgcc_s.so.1()(64bit) libgcc_s.so.1(GCC_3.0)(64bit) libm.so.6()
(64bit) libm.so.6(GLIBC_2.2.5)(64bit) libpthread.so.0()(64bit)
libpthread.so.0(GLIBC_2.2.5)(64bit) libpthread.so.0(GLIBC_2.3.2)(64bit)
librt.so.1()(64bit) librt.so.1(GLIBC_2.2.5)(64bit) libstdc++.so.6()(64bit)
libstdc++.so.6(CXXABI_1.3)(64bit) libstdc++.so.6(GLIBCXX_3.4)(64bit) rtld(GNU_HASH)
Checking for unpackaged file(s): /usr/lib64/rpm/check-files
/usr/lib64/rpm/../../../var/tmp/iperf-root
Wrote: /usr/src/rpm/SRPMS/iperf-2.0.9-XR_6.1.1.src.rpm
Wrote: /usr/src/rpm/RPMS/x86_64/iperf-2.0.9-XR_6.1.1.x86_64.rpm
...

localhost:~$ ls -l /usr/src/rpm/RPMS/x86_64/
total 48
-rw-r--r-- 1 root root 48118 Sep 13 02:03 iperf-2.0.9-XR_6.1.1.x86_64.rpm
```

9. Transfer the RPM file to XR.

a. Note down the port number on XR for transferring the RPM file.

```
localhost:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant port rtr
The forwarded ports for the machine are listed below. Please note that
these values may differ from values configured in the Vagrantfile if the
provider supports automatic port collision detection and resolution.

    22 (guest) => 2223 (host)
 57722 (guest) => 2222 (host)
```

b. Access the WRL7 build instance, and copy the RPM file by using the SCP command with the port number of XR.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant ssh wrl7_build
Last login: Tue Sep 13 01:49:37 2016 from 10.0.2.2

localhost:~$ scp -P 2222 /usr/src/rpm/RPMS/x86_64/iperf-2.0.9-XR_6.1.1.x86_64.rpm
vagrant@10.0.2.2:/home/vagrant/
vagrant@10.0.2.2's password:
iperf-2.0.9-XR_6.1.1.x86_64.rpm
```

10. Install the application (iPerf) on XR.

a. Access XR through SSH.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant ssh rtr
Last login: Fri Sep  9 19:20:56 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$
```

b. Verify the presence of the RPM file on XR.

```
xr-vm_node0_RP0_CPU0:~$ ls -l iperf-2.0.9-XR_6.1.1.x86_64.rpm
-rw-r--r-- 1 vagrant vagrant 48118 Sep 13 06:33 iperf-2.0.9-XR_6.1.1.x86_64.rpm
```

c. Install iPerf by using yum.

```
xr-vm_node0_RP0_CPU0:~$ sudo yum install -y  iperf-2.0.9-XR_6.1.1.x86_64.rpm
Loaded plugins: downloadonly, protect-packages, rpm-persistence
Setting up Install Process
Examining iperf-2.0.9-XR_6.1.1.x86_64.rpm: iperf-2.0.9-XR_6.1.1.x86_64
Marking iperf-2.0.9-XR_6.1.1.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package iperf.x86_64 0:2.0.9-XR_6.1.1 will be installed
--> Finished Dependency Resolution

...

Total size: 103 k
Installed size: 103 k
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : iperf-2.0.9-XR_6.1.1.x86_64

Installed:
```

```
  iperf.x86_64 0:2.0.9-XR_6.1.1


Complete!
xr-vm_node0_RP0_CPU0:~$
```

d. Verify iPerf installation.

```
xr-vm_node0_RP0_CPU0:~$ iperf -v
iperf version 2.0.9 (1 June 2016) pthreads
```

11. Test the natively installed application (iPerf) on XR.

a. Access the XR router console and configure the Third-party Application (TPA) access for outside networks.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:

RP/0/RP0/CPU0:ios# config
Tue Sep 13 06:46:56.368 UTC
RP/0/RP0/CPU0:ios(config)# tpa address-family ipv4 update-source loopback 0
RP/0/RP0/CPU0:ios(config)# commit
Tue Sep 13 06:47:04.642 UTC
RP/0/RP0/CPU0:ios(config)# end
RP/0/RP0/CPU0:ios# bash -c ip route
Tue Sep 13 06:47:43.792 UTC
default dev fwdintf  scope link  src 1.1.1.1
10.0.2.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 10.0.2.15
```

b. Exit the XR router console, and launch the iPerf server on XR.

```
RP/0/RP0/CPU0:ios# exit
Connection to localhost closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant ssh rtr
Last login: Tue Sep 13 06:44:53 2016 from 10.0.2.2

xr-vm_node0_RP0_CPU0:~$ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 64.0 MByte (default)
------------------------------------------------------------
```

12. Install the iPerf (client) on `devbox`.

a. Access `devbox` through SSH.

```
xr-vm_node0_RP0_CPU0:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/native-app-topo-bootstrap (master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-92-generic x86_64)
...
```

13. Install iPerf application.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get -y install iperf
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  iperf
...
```

14. Test the iPerf application on `devbox`.

    a. Configure TPA route to XR from `devbox`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo ip route add 1.1.1.1/32 via 11.1.1.10
vagrant@vagrant-ubuntu-trusty-64:~$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=255 time=15.1 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=255 time=3.81 ms
^C
--- 1.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 3.817/9.480/15.143/5.663 ms
```

    b. Test if the iPerf client on `devbox` can communicate with the iPerf server on XR.

```
vagrant@vagrant-ubuntu-trusty-64:~$ iperf -c 1.1.1.1 -u
------------------------------------------------------------
Client connecting to 1.1.1.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 11.1.1.20 port 34348 connected with 1.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.256 ms    0/  893 (0%)
```

You have successfully built an application RPM and hosted it natively by using vagrant.

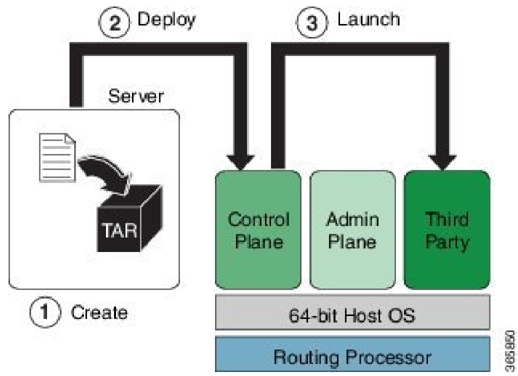
Hosting an Application within a Linux Container (LXC) by Using Vagrant

This section describes how you can host an application within your own Linux container (LXC) by using vagrant.

Workflow for Deploying Your LXC Container

The workflow for launching your container on IOS XR is described in this section and illustrated in the following topology.

Figure 8: LXC Container Deployment Workflow



1. Build the container `rootfs` tar ball on `devbox`.
2. Transfer the `rootfs` tar ball to IOS XR (`rtr`).
3. Launch the `rootfs` by running the **virsh** command.

Procedure

To host your application within your own container, use the following steps.

> **Note** Ensure you have created an application development topology as described in Setting up an Application Development Topology By Using Vagrant, on page 19, before proceeding with the following steps.

1. Navigate to the `lxc-app-topo-bootstrap` directory and ensure the vagrant instance is running. If not, launch the vagrant instance.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant status
Current machine states:

rtr                          aborted (virtualbox)
devbox                       aborted (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant up
Bringing machine 'rtr' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
==> rtr: Clearing any previously set forwarded ports...
==> rtr: Clearing any previously set network interfaces...
==> rtr: Preparing network interfaces based on configuration...
    rtr: Adapter 1: nat
    rtr: Adapter 2: intnet
==> rtr: Forwarding ports...
    rtr: 57722 (guest) => 2222 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
==> rtr: Running 'pre-boot' VM customizations...
==> rtr: Booting VM...
==> rtr: Waiting for machine to boot. This may take a few minutes...
```

```
    rtr: SSH address: 127.0.0.1:2222
    rtr: SSH username: vagrant
    rtr: SSH auth method: private key
    rtr: Warning: Remote connection disconnect. Retrying...
    ...
==> rtr: Machine booted and ready!
...
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: flag to force provisioning. Provisioners marked to run always will still run.
==> devbox: Checking if box 'ubuntu/trusty64' is up to date...
==> devbox: A newer version of the box 'ubuntu/trusty64' is available! You currently
==> devbox: have version '20160801.0.0'. The latest is version '20160826.0.1'. Run
==> devbox: `vagrant box update` to update.
==> devbox: Clearing any previously set forwarded ports...
==> devbox: Fixed port collision for 22 => 2222. Now on port 2200.
==> devbox: Clearing any previously set network interfaces...
==> devbox: Preparing network interfaces based on configuration...
    devbox: Adapter 1: nat
    devbox: Adapter 2: intnet
==> devbox: Forwarding ports...
    devbox: 22 (guest) => 2200 (host) (adapter 1)
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2200
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    devbox: Warning: Remote connection disconnect. Retrying...
    devbox: Warning: Remote connection disconnect. Retrying...
==> devbox: Machine booted and ready!
...
    devbox: Guest Additions Version: 4.3.36
    devbox: VirtualBox Version: 5.0
==> devbox: Configuring and enabling network interfaces...
==> devbox: Mounting shared folders...
    devbox: /vagrant => C:/Users/annseque/vagrant-xrdocs/lxc-app-topo-bootstrap
==> devbox: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> devbox: flag to force provisioning. Provisioners marked to run always will still
run.

==> rtr: Machine 'rtr' has a post `vagrant up` message. This is a message
==> rtr: from the creator of the Vagrantfile, and not from Vagrant itself:
==> rtr:
==> rtr:
==> rtr:     Welcome to the IOS XRv (64-bit) Virtualbox.
==> rtr:     To connect to the XR Linux shell, use: 'vagrant ssh'.
==> rtr:     To ssh to the XR Console, use: 'vagrant port' (vagrant version > 1.8)
==> rtr:     to determine the port that maps to guestport 22,
==> rtr:     then: 'ssh vagrant@localhost -p <forwarded port>'
...
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant status
Current machine states:


rtr                       running (virtualbox)
devbox                    running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

2. Access the `devbox` through SSH and install LXC tools.

   To launch an LXC container, you need the following, which can be obtained by installing LXC tools:

- A container rootfs tar ball
- An XML file to launch the container using **virsh/libvirt**

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Thu Sep  1 03:55:29 UTC 2016

  System load:  0.99              Processes:           94
  Usage of /:   3.9% of 39.34GB   Users logged in:     0
  Memory usage: 14%               IP address for eth0: 10.0.2.15
  Swap usage:   0%                IP address for eth1: 11.1.1.20

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

25 packages can be updated.
12 updates are security updates.

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

-------------------------------------------------------------------------------------
Last login: Wed Aug 31 04:02:20 2016 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get update
Ign http://archive.ubuntu.com trusty InRelease
Get:1 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
...
Get:33 http://archive.ubuntu.com trusty-backports/universe Translation-en [36.8 kB]
Hit http://archive.ubuntu.com trusty Release
...
Hit http://archive.ubuntu.com trusty/universe Translation-en
Ign http://archive.ubuntu.com trusty/main Translation-en_US
Ign http://archive.ubuntu.com trusty/multiverse Translation-en_US
Ign http://archive.ubuntu.com trusty/restricted Translation-en_US
Ign http://archive.ubuntu.com trusty/universe Translation-en_US
Fetched 4,022 kB in 16s (246 kB/s)
Reading package lists... Done

-------------------------------------------------------------------------------------
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get -y install lxc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bridge-utils cgmanager cloud-image-utils debootstrap dnsmasq-base euca2ools
  genisoimage libaio1 libboost-system1.54.0 libboost-thread1.54.0 liblxc1
  libmnl0 libnetfilter-conntrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbd1 libseccomp2 libxslt1.1 lxc-templates python-distro-info python-lxml
  python-requestbuilder python-setuptools python3-lxc qemu-utils sharutils
  uidmap
Suggested packages:
  cgmanager-utils wodim cdrkit-doc btrfs-tools lvm2 lxctl qemu-user-static
  python-lxml-dbg bsd-mailx mailx
The following NEW packages will be installed:
  bridge-utils cgmanager cloud-image-utils debootstrap dnsmasq-base euca2ools
  genisoimage libaio1 libboost-system1.54.0 libboost-thread1.54.0 liblxc1
```

```
  libmnl0 libnetfilter-conntrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbd1 libseccomp2 libxslt1.1 lxc lxc-templates python-distro-info
  python-lxml python-requestbuilder python-setuptools python3-lxc qemu-utils
  sharutils uidmap
0 upgraded, 30 newly installed, 0 to remove and 52 not upgraded.
Need to get 6,469 kB of archives.
After this operation, 25.5 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main libaio1 amd64 0.3.109-4 [6,364 B]
...
Get:30 http://archive.ubuntu.com/ubuntu/ trusty-updates/main debootstrap all
1.0.59ubuntu0.5 [29.6 kB]
Fetched 6,469 kB in 22s (289 kB/s)
Selecting previously unselected package libaio1:amd64.
(Reading database ... 62989 files and directories currently installed.)
Preparing to unpack .../libaio1_0.3.109-4_amd64.deb ...
...
Setting up lxc (1.0.8-0ubuntu0.3) ...
lxc start/running
Setting up lxc dnsmasq configuration.
Processing triggers for ureadahead (0.100.0-16) ...
Setting up lxc-templates (1.0.8-0ubuntu0.3) ...
Setting up libnss3-nssdb (2:3.23-0ubuntu0.14.04.1) ...
Setting up libnss3:amd64 (2:3.23-0ubuntu0.14.04.1) ...
Setting up librados2 (0.80.11-0ubuntu1.14.04.1) ...
Setting up librbd1 (0.80.11-0ubuntu1.14.04.1) ...
Setting up qemu-utils (2.0.0+dfsg-2ubuntu1.27) ...
Setting up cloud-image-utils (0.27-0ubuntu9.2) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
```

3. Verify that the LXC was properly installed.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-start --version
1.0.8
```

4. Create the LXC container with a standard Ubuntu base template and launch it in `devbox`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-create -t ubuntu --name xr-lxc-app
Checking cache download in /var/cache/lxc/trusty/rootfs-amd64 ...
Installing packages in template: ssh,vim,language-pack-en
Downloading ubuntu trusty minimal ...
I: Retrieving Release
I: Retrieving Release.gpg
...
Generation complete.
Setting up perl-modules (5.18.2-2ubuntu1.1) ...
Setting up perl (5.18.2-2ubuntu1.1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...
Processing triggers for initramfs-tools (0.103ubuntu4.4) ...
Download complete
Copy /var/cache/lxc/trusty/rootfs-amd64 to /var/lib/lxc/xr-lxc-app/rootfs ...
Copying rootfs to /var/lib/lxc/xr-lxc-app/rootfs ...
Generating locales...
  en_US.UTF-8... up-to-date
Generation complete.
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
Creating SSH2 ECDSA key; this may take some time ...
Creating SSH2 ED25519 key; this may take some time ...
update-rc.d: warning: default stop runlevel arguments (0 1 6) do not match ssh
Default-Stop values (none)
invoke-rc.d: policy-rc.d denied execution of start.

Current default time zone: 'Etc/UTC'
Local time is now:      Thu Sep  1 04:46:22 UTC 2016.
```

```
Universal Time is now:  Thu Sep  1 04:46:22 UTC 2016.


##
# The default user is 'ubuntu' with password 'ubuntu'!
# Use the 'sudo' command to run tasks as root in the container.
##
```

5. Verify if the LXC container has been successfully created.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-ls --fancy
NAME        STATE    IPV4  IPV6  AUTOSTART
------------------------------------------
xr-lxc-app  STOPPED  -     -     NO
```

6. Start the LXC container.

   You will be prompted to log into the LXC container. The login credentials are **ubuntu/ubuntu**.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-start --name xr-lxc-app
<4>init: plymouth-upstart-bridge main process (5) terminated with status 1
...

xr-lxc-app login: ubuntu
Password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ubuntu@xr-lxc-app:~$
```

7. Install your application within the LXC container.

   For the sake of illustration, in this example we will install the iPerf application.

```
ubuntu@xr-lxc-app:~$ sudo apt-get -y install iperf
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  iperf
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 56.3 kB of archives.
After this operation, 174 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/universe iperf amd64 2.0.5-3 [56.3 kB]
Fetched 56.3 kB in 16s (3,460 B/s)
Selecting previously unselected package iperf.
(Reading database ... 14648 files and directories currently installed.)
Preparing to unpack .../iperf_2.0.5-3_amd64.deb ...
Unpacking iperf (2.0.5-3) ...
Setting up iperf (2.0.5-3) ...
ubuntu@xr-lxc-app:~$
```

8. Change the SSH port inside the container and verify that it has been correctly assigned.

   When you deploy your container to IOS XR, it shares the network namespace with XR. Since IOS XR already uses Ports 22 and 57722 for other purposes, you must pick some other port number for your container.

```
ubuntu@xr-lxc-app:~$ sudo sed -i s/Port\ 22/Port\ 58822/ /etc/ssh/sshd_config
[sudo] password for ubuntu:

ubuntu@xr-lxc-app:~$ cat /etc/ssh/sshd_config | grep Port
Port 58822
ubuntu@xr-lxc-app:~$
```

9. Shut the container down.

```
ubuntu@xr-lxc-app:~$ sudo shutdown -h now
ubuntu@xr-lxc-app:~$
Broadcast message from ubuntu@xr-lxc-app
        (/dev/lxc/console) at 5:17 ...

The system is going down for halt NOW!
<4>init: tty4 main process (369) killed by TERM signal
...
wait-for-state stop/waiting
 * Asking all remaining processes to terminate...
   ...done.
 * All processes ended within 1 seconds...
   ...done.
 * Deactivating swap...
   ...done.
mount: cannot mount block device /dev/sda1 read-only
 * Will now halt
```

10. Assume the root user role.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo -s
root@vagrant-ubuntu-trusty-64:~# whoami
root
```

11. Navigate to the `/var/lib/lxc/xr-lxc-app/` directory and package the `rootfs` into a tar ball.

```
root@vagrant-ubuntu-trusty-64:~# cd /var/lib/lxc/xr-lxc-app/
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app# ls
config  fstab  rootfs
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app# cd rootfs
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs# tar -czvf
xr-lxc-app-rootfs.tar.gz *
tar: dev/log: socket ignored
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs#
```

12. Transfer the `rootfs` tar ball to the home directory (`~/` or `/home/vagrant`) and verify if the transfer is successful.

```
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs# mv *.tar.gz /home/vagrant
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs# ls -l /home/vagrant
total 120516
-rw-r--r-- 1 root root 123404860 Sep  1 05:22 xr-lxc-app-rootfs.tar.gz
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs#
```

13. Create an LXC spec XML file for specifying attributes required to launch the LXC container with the application.

    You must navigate to the `/home/vagrant` directory on `devbox` and use a vi editor to create the XML file. Save the file as `xr-lxc-app.xml`.

    A sample LXC spec file to launch the application within the container is as shown.

```
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs# exit
exit
vagrant@vagrant-ubuntu-trusty-64:~$ pwd
/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~$ vi xr-lxc-app.xml

-------------------------------------------------------------------------------------
<domain type='lxc' xmlns:lxc='http://libvirt.org/schemas/domain/lxc/1.0' >
<name>xr-lxc-app</name>
<memory>327680</memory>
<os>
<type>exe</type>
<init>/sbin/init</init>
</os>
<lxc:namespace>
<sharenet type='netns' value='global-vrf'/>
</lxc:namespace>
<vcpu>1</vcpu>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
<emulator>/usr/lib64/libvirt/libvirt_lxc</emulator>
<filesystem type='mount'>
<source dir='/misc/app_host/xr-lxc-app/'/>
<target dir='/'/>
</filesystem>
<console type='pty'/>
</devices>
</domain>
```

    In IOS-XR the `global-vrf` network namespace contains all the XR GigE or management interfaces. The `sharenet` configuration in the XML file ensures that the container on being launched has native access to all XR interfaces.

    `/misc/app_host/` on IOS XR is a special mount volume that is designed to provide nearly 3.9GB of disk space. This mount volume can be used to host custom container `rootfs` and other large files without occupying disk space on XR. In this example, we expect to untar the `rootfs` to the `/misc/app_host/xr-lxc-app/` directory.

14. Verify if the `rootfs` tar ball and the LXC XML spec file are present in the home directory.

```
root@vagrant-ubuntu-trusty-64:~# pwd
/home/vagrant
root@vagrant-ubuntu-trusty-64:~# ls -l
total 119988
-rw-r--r-- 1 root root 122863332 Jun 16 19:41 xr-lxc-app-rootfs.tar.gz
-rw-r--r-- 1 root root       590 Jun 16 23:29 xr-lxc-app.xml
root@vagrant-ubuntu-trusty-64:~#
```

15. Transfer the `rootfs` tar ball and XML spec file to XR.

There are two ways of transferring the files: Through the GigE interface (a little slower) or the management interface. You can use the method that works best for you.

- **Transfer Through the Management Interface of XR:**

a. Check the port number that maps to the management port on XR.

Vagrant forwards the port number 57722 to a host port for XR over the management port. In a virtual box, the IP address of the host (your laptop) is always 10.0.2.2 for the port that was translated (NAT).

```
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant port rtr
The forwarded ports for the machine are listed below. Please note that
these values may differ from values configured in the Vagrantfile if the
provider supports automatic port collision detection and resolution.

    22 (guest) => 2223 (host)
 57722 (guest) => 2222 (host)
```

The output shows that port number 2222 maps to port number 57722.

b. Access `devbox` and use the port number 2222 to transfer the `rootfs` tar ball and XML spec file to XR.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Sep  2 05:38:20 UTC 2016

  System load:  0.49              Users logged in:       0
  Usage of /:   6.4% of 39.34GB   IP address for eth0:   10.0.2.15
  Memory usage: 25%               IP address for eth1:   11.1.1.20
  Swap usage:   0%                IP address for lxcbr0: 10.0.3.1
  Processes:    80

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Fri Sep  2 05:38:20 2016 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-64:~$ scp -P 2222 /home/vagrant/*.*
vagrant@10.0.2.2:/misc/app_host/scratch
The authenticity of host '[10.0.2.2]:2222 ([10.0.2.2]:2222)' can't be
established.
ECDSA key fingerprint is db:25:e2:27:49:2a:7b:27:e1:76:a6:7a:e4:70:f5:f7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.0.2.2]:2222' (ECDSA) to the list of known hosts.
vagrant@10.0.2.2's password:
xr-lxc-app-rootfs.tar.gz
```

```
100%  234MB  18.0MB/s   00:13
xr-lxc-app.xml

100%  591     0.6KB/s   00:00
vagrant@vagrant-ubuntu-trusty-64:~$
```

- **Transfer Through the GigE Interface of XR:**

a. Determine the GigE interface IP address on XR.

```
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh rtr
Last login: Wed Aug 31 07:09:51 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 08:00:27:5a:29:77
          inet addr:11.1.1.10  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe5a:2977/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:3 overruns:0 carrier:1
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:42 (42.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 08:00:27:13:ad:eb
          inet addr:10.0.2.15  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe13:adeb/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:94 errors:0 dropped:0 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:1
          collisions:0 txqueuelen:1000
          RX bytes:13325 (13.0 KiB)  TX bytes:11041 (10.7 KiB)

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:1496  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:302 (302.0 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)
```

In this example, the IP address of the GigE interface is 11.1.1.10.

b. Copy the `rootfs` tar ball to XR by using the GigE interface address.

```
vagrant@vagrant-ubuntu-trusty-64:~$ scp -P 57722
/home/vagrant/xr-lxc-app-rootfs.tar.gz
vagrant@11.1.1.10:/misc/app_host/scratch/
The authenticity of host '[11.1.1.10]:57722 ([11.1.1.10]:57722)' can't be
established.
ECDSA key fingerprint is db:25:e2:27:49:2a:7b:27:e1:76:a6:7a:e4:70:f5:f7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[11.1.1.10]:57722' (ECDSA) to the list of known
hosts.
vagrant@11.1.1.10's password:
xr-lxc-app-rootfs.tar.gz
```

c. Copy the XML spec file to XR by using the GigE interface address.

```
vagrant@vagrant-ubuntu-trusty-64:~$ scp -P 57722 /home/vagrant/xr-lxc-app.xml
vagrant@11.1.1.10:/misc/app_host/scratch/
vagrant@11.1.1.10's password:
xr-lxc-app.xml
```

16. Create a directory (`/misc/app_host/xr-lxc-app/`)on XR (`rtr`) to untar the `rootfs` tar ball.

```
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh rtr
Last login: Fri Sep  2 05:49:01 2016 from 10.0.2.2

xr-vm_node0_RP0_CPU0:~$ sudo mkdir /misc/app_host/xr-lxc-app/
```

17. Navigate to the `/misc/app_host/xr-lxc-app/` directory and untar the tar ball.

```
xr-vm_node0_RP0_CPU0:~$ cd /misc/app_host/xr-lxc-app/
xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ sudo tar -zxf
../scratch/xr-lxc-app-rootfs.tar.gz
tar: dev/audio3: Cannot mknod: Operation not permitted
...
```

18. Use the XML spec file to launch the container and verify its existence on XR.

```
xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ virsh create
/misc/app_host/scratch/xr-lxc-app.xml
Domain xr-lxc-app created from /misc/app_host/scratch/xr-lxc-app.xml

xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ virsh list
 Id    Name                           State
----------------------------------------------------
 2095  xr-lxc-app                     running
 4932  sysadmin                       running
 12086 default-sdr--1                 running
```

19. Log into the container. The default login credentials are `ubuntu/ubuntu`.

There are two ways of logging into the container. You can use the method that works best for you:

- **Logging into the container by using virsh command:**

```
xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ virsh console xr-lxc-app
Connected to domain xr-lxc-app
Escape character is ^]
```

```
init: Unable to create device: /dev/kmsg
* Stopping Send an event to indicate plymouth is up                    [ OK ]
* Starting Mount filesystems on boot                                   [ OK ]
* Starting Signal sysvinit that the rootfs is mounted                  [ OK ]
* Starting Fix-up sensitive /proc filesystem entries                   [ OK ]

xr-lxc-app login:  * Starting OpenSSH server                            [ OK ]


Ubuntu 14.04.5 LTS xr-lxc-app tty1
xr-lxc-app login: ubuntu
Password:
Last login: Fri Sep  2 05:40:11 UTC 2016 on lxc/console
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.14.23-WR7.0.0.2_standard x86_64)

 * Documentation:  https://help.ubuntu.com/
ubuntu@xr-lxc-app:~$
```

- **Logging into the container by using SSH:**

  Use the SSH port number you configured, 58822, and any of XR interface IP addresses to log in.

```
xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ ssh -p 58822 ubuntu@11.1.1.10
Warning: Permanently added '[11.1.1.10]:58822' (ECDSA) to the list of known hosts.
ubuntu@11.1.1.10's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.14.23-WR7.0.0.2_standard x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Sep  2 07:42:37 2016
ubuntu@xr-lxc-app:~$
```

Note

- To exit the container, use the press **CTRL** and **]** keys simultaneously.
- To access the container directly from your host machine, ensure you forward the intended port (in this example, 58822) to your laptop (any port of your choice), in the Vagrant file:

```
node.vm.network "forwarded_port", guest: 58822, host: 58822
```

  You can then SSH to the LXC container by using the following command:

```
ssh -p 58822 vagrant@localhost
```

20. Verify if the interfaces on XR are available inside the LXC container.

The LXC container operates as your own Linux server on XR. Because the network namespace is shared between the LXC and XR, all of XR interfaces (GigE, management, and so on) are available to bind to and run your applications.

```
ubuntu@xr-lxc-app:~$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 08:00:27:5a:29:77
          inet addr:11.1.1.10  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe5a:2977/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:186070 errors:0 dropped:0 overruns:0 frame:0
          TX packets:155519 errors:0 dropped:3 overruns:0 carrier:1
          collisions:0 txqueuelen:1000
          RX bytes:301968784 (301.9 MB)  TX bytes:10762900 (10.7 MB)
```

```
Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 08:00:27:13:ad:eb
          inet addr:10.0.2.15  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe13:adeb/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:170562 errors:0 dropped:0 overruns:0 frame:0
          TX packets:70309 errors:0 dropped:0 overruns:0 carrier:1
          collisions:0 txqueuelen:1000
          RX bytes:254586763 (254.5 MB)  TX bytes:3886846 (3.8 MB)

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:1496  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:155549 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:10765764 (10.7 MB)

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:64 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9400 (9.4 KB)  TX bytes:9400 (9.4 KB)
```

21. Configure the container to communicate outside XR with other nodes in the network.

By default, the IOS-XRv vagrant box is set up to talk to the internet using a default route through your management port. If you want the router to use the routing table to talk to other nodes in the network, then you must configure **tpa-address**. This becomes the **src-hint** for all Linux application traffic.

In this example, we use Loopback 0 for **tpa-address** to ensure that the IP address for any originating traffic for applications on the XR is a reachable IP address across your topology.

```
ubuntu@xr-lxc-app:~$ exit
logout
Connection to 11.1.1.10 closed.
xr-vm_node0_RP0_CPU0:/misc/app_host/xr-lxc-app$ exit
logout
Connection to 127.0.0.1 closed.

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$  vagrant port rtr | grep 22
    22 (guest) => 2223 (host)
 57722 (guest) => 2222 (host)

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:


RP/0/RP0/CPU0:ios# configure
Fri Sep  2 08:03:05.094 UTC
RP/0/RP0/CPU0:ios(config)# interface loopback 0
```

```
RP/0/RP0/CPU0:ios(config-if)# ip address 1.1.1.1/32
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# tpa address-family ipv4 update-source loopback 0
RP/0/RP0/CPU0:ios(config)# commit
Fri Sep  2 08:03:39.602 UTC
RP/0/RP0/CPU0:ios(config)# exit
RP/0/RP0/CPU0:ios# bash
Fri Sep  2 08:03:58.232 UTC

[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 1.1.1.1
10.0.2.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 10.0.2.15
```

You can see the configured Loopback 0 IP address (1.1.1.1).

22. Test your application within the launched container.

We installed iPerf in our container. We will run the iPerf server within the container, and the iPerf client on the `devbox` and see if they can communicate. Basically, the hosted application within a container on `rtr` should be able to talk to a client application on `devbox`.

a. Check if the iPerf server is running within the LXC container on XR.

```
[xr-vm_node0_RP0_CPU0:~]$ssh -p 58822 ubuntu@11.1.1.10
Warning: Permanently added '[11.1.1.10]:58822' (ECDSA) to the list of known hosts.
ubuntu@11.1.1.10's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.14.23-WR7.0.0.2_standard x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Fri Sep  2 07:47:28 2016 from 11.1.1.10

ubuntu@xr-lxc-app:~$ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 64.0 MByte (default)
------------------------------------------------------------
```

b. Check if XR Loopback interface is accessible on devbox. (Open a new Git bash window for this step.)

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ cd vagrant-xrdocs

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd lxc-app-topo-bootstrap/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/lxc-app-topo-bootstrap (master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Sep  2 05:51:19 UTC 2016

  System load:  0.08              Users logged in:       0
  Usage of /:   6.4% of 39.34GB   IP address for eth0:   10.0.2.15
  Memory usage: 28%               IP address for eth1:   11.1.1.20
  Swap usage:   0%                IP address for lxcbr0: 10.0.3.1
  Processes:    77

  Graph this data and manage this system at:
    https://landscape.canonical.com/
```

```
  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

53 packages can be updated.
26 updates are security updates.

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Fri Sep  2 05:51:21 2016 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-64:~$ sudo ip route add 1.1.1.1/32 via 11.1.1.10
vagrant@vagrant-ubuntu-trusty-64:~$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=255 time=1.87 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=255 time=10.5 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=255 time=4.13 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 1.876/5.510/10.520/3.661 ms
```

c. Install the iPerf client on `devbox`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get install iperf
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  iperf
0 upgraded, 1 newly installed, 0 to remove and 52 not upgraded.
Need to get 56.3 kB of archives.
After this operation, 174 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/universe iperf amd64 2.0.5-3 [56.3
kB]
Fetched 56.3 kB in 10s (5,520 B/s)
Selecting previously unselected package iperf.
(Reading database ... 64313 files and directories currently installed.)
Preparing to unpack .../iperf_2.0.5-3_amd64.deb ...
Unpacking iperf (2.0.5-3) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up iperf (2.0.5-3) ...
```

d. Launch the iPerf client on `devbox` and verify if it is communicating with the iPerf server within the LXC on XR.

```
vagrant@vagrant-ubuntu-trusty-64:~$ iperf -u -c 1.1.1.1
------------------------------------------------------------
Client connecting to 1.1.1.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 11.1.1.20 port 37800 connected with 1.1.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   1.791 ms    0/  893 (0%)
```

You have successfully hosted an application within a Linux container by using vagrant.

Installing Docker on Cisco IOS XR By Using Vagrant

This section describes how you can install a Docker container on Cisco IOS XR by using Vagrant.

Setup Options for Dockers on XR

You can choose any of the following setups for using Dockers on XR.

- **Public Docker-Hub registry**: You can configure a public Docker-Hub with the correct DNS resolution so that it is accessible to all users. This is the simplest form of Docker setup.
- **Private Docker-Hub unsecured registry**: You can configure a private Docker-Hub registry without security, if you are planning to run the registry inside a secured part of your network.
- **Private Docker-Hub self-signed registry**: You can configure a private Docker-Hub registry enabled with TLS. This is more secure than using a local unsecured registry.
- **Private Docker-Hub secured registry**: You can configure a private Docker-Hub secured registry, created using a certificate obtained from a Certificate Authority (CA) server. The steps used to set this up are identical to a private Docker-Hub self-signed registry except for the creation of the certificate.
- **Tarball image/container**: You can create and configure a Docker container on your laptop and package it as an image or a container tar ball. You can then transfer the tar ball to XR, and extract the Docker container for use.

For information on implementing these setup options, see the XR toolbox, Part 6: Running Docker Containers on IOS-XR (6.1.2+) section on Github.

Secure Onboarding of Signed Third-Party Applications

Table 6: Feature History Table

Feature Name	Release Information	Feature Description
Secure Onboarding of Signed Third-Party Applications	Release 7.10.1	Introduced in this release on: NCS 5500 fixed port routers Cisco IOS XR now supports onboarding signed (authenticated) third-party (non-native Cisco IOS XR) applications onto the XR routers securely as per Cisco policies and standards. Earlier you could onboard only signed Cisco IOS XR native images and RPMs onto the router.

Cisco IOS XR now supports onboarding signed third-party (non-native IOS XR) applications onto the XR routers. The signed third-party applications (TPA) must be in the form of a docker image in Release 7.10.1, and these applications are onboarded through RPMs.

RPM database consists of GNU Privacy Guard (GPG) keys. The GPG keys are used to validate the signatures of the signed TPA. All TPA RPMs must be signed, and, for security reasons, their signatures are verified before they are installed on the XR system.

Prerequisites

Ensure that your router supports the Secure Zero Touch Provisioning (SZTP), which is based on RFC 8572.

Key Terms

Owner Certificate: The owner certificate (OC) is an X.509 certificate [RFC8572] that is used to identify an owner, for example, an organization. The OC can be signed by any certificate authority (CA). The public key in OC is used to verify CA signature of the device, Signed Conveyed Information (CI or CIA), and to verify signed JSON config files and signed Key Packages. The OC structure must contain the owner certificate itself, as well as all intermediate certificates leading to the pinned-domain-cert (PDC) certificate specified in the ownership voucher.

Ownership Voucher: The ownership voucher (OV) [RFC8366] is used to securely identify the device's owner, as known to the manufacturer. OV is signed by customer provided key certification and once it is authenticated, the PDC node is extracted to verify OC. The OV is used to verify that the owner certificate has a chain of trust leading to the trusted Pinned Domain Cert certificate (PDC), which is a pinned X.509 cert from the CA, included in the ownership voucher. OVs are issued by Cisco's Manufacturer Authorized Signing Authority (MASA) service. For information on MASA, see the Manufacturer Authorized Signing Authority (MASA) chapter from System Security Configuration Guide for Cisco 8000 Series Routers. OV has PID/Serial number (SN) and has expiry date or nonce.

Secure **U**nique **D**evice **I**dentifier (SUDI): It is a unique ID per-device certificate (based on IEEE 802.1AR) programmed into the TAm chip during the device manufacturing. It is unique per card (one per RP, LC, and so on). It includes Product Identification (PID) and Serial number (SN) of device. It is signed by Cisco for proof of authenticity

- **Product Identification:** Each router is given a distinct product identification (PID) number, which is the equivalent to a stock-keeping unit (SKU) number.
- **Serial Number:** The serial number (SN) of the router is typically in the format of LLLYYWWSSSS. LLL represents the location of manufacturing.YYand WW represent the year and week of manufacture respectively. SSSS is the unique code of your router. You can find the serial number at the bottom of the router or by running the show version command.

How Can I Onboard My Applications Securely?

To securely onboard your application, you must:

- Establish Device Ownership, on page 48
- Generate KeyPackage, on page 48
- Onboard Key Package on Router, on page 52
- Generate Signed RPM, on page 55
- Onboard Signed RPM Package on Router, on page 57

Establish Device Ownership

For the details of device ownership establishment, see Establish Device Ownership section from the *System Security Configuration Guide for Cisco NCS 540 Series Routers* Guide.

Once the ownership is established, it is stored in Trusted Anchor mode (TAm) of the router. The ownership information is persistent between device boot ups and factory reset.

SUDI-based authentication and validation of the device is also possible. For more details, contact Cisco Technical Assistance.

Generate KeyPackage

Key package is a Cryptographic Message Syntax (CMS [RFC5652]) file that has a payload and must be digitally signed with private keys of the customer's Ownership Certificate (OC).

The payload of the tar file contains:

- Customer Keys (X509 or GPG), on page 50
- Key Package Configuration File, on page 51

This tar file is embedded in the CMS envelope and digitally signed with private keys of the Customer Ownership Certificate.

You can pack several key packages along with a configuration file, into a single bundle and install the bundle at once, by creating a key package bundle. This bundle must be signed by the device OC, else installation of the bundle or individual key packages fails at the verification.

Note The key package is used to onboard public keys only. Private keys should NOT be onboarded through the key package.

The following restrictions apply to key package infrastructure:

- Supports only a single key in a single key package.
- The accepted time stamp range is years 2000—2100.

Create the Keys

The Github repository provides commands to perform key request of different types such as ADD, DELETE.

Note Once a key package is onboarded into a router, we cannot roll back or undo the operation. If a key is added/deleted/revoked through a key package, the operation cannot be undone or rolled back. If you want to go back to the previous state of keys, you must create a new key package.

For more details on creating the GPG keys, see Customer Keys (X509 or GPG), on page 50.

Update the Keys

A new router image has new ISO and a new key package. The old key package is replaced with the new package.

If the key to be revoked is present in the ALLOWED_LIST, you must:

1. Uninstall (automated or manual) the older RPMs that were signed with the revoked key.
2. Add the key to REVOKED_LIST through another key package.

If the key to be revoked does not exist in ALLOWED_LIST, you must:

- Generate a key package to add the key to REVOKED_LIST.

To delete a key, you must create a key package with the delete option, and must package it as a GISO.

Note To prevent reuse of key package on the same system, a TIMESTAMP field is present in the config.txt file of the key package. When a key package is generated, the config.txt file should contain TIMESTAMP=<time at which the key package is generated>. This time mst be in RFC 2822 format.

Error messages

Based on the key package error type, the following syslog or error messages are invoked:

Table 7: Key Package Error Messages

Message	Description
KPKG_INIT_FAIL	Failed to initialize key package.
KPKG_VALIDATION_FAIL	Key package signature validation failed. Check the keys used for signing the key package or the tampering of the file.
KPKG_CONFIG_INVALID	Errors or inconsistencies in the configuration file of the key package. Check for the validity of the key package configuration file.
KPKG_KEYSIZE_ERR	Installed Key is more than maximum limit. Default maximum key size is 3 KB.
KPKG_REVOKED_INSTALL_ERR	An attempt to install a revoked key which is not allowed.
KPKG_KEY_MISSING_ERR	Key is not found in the key package.
KPKG_USAGE_LEN_EXCEEDED	Ensure the length of USAGE is within the limit of six characters.
KPKG_OPTNAL_LEN_ERR	Length of the optional string exceeded the limit.
KPKG_TIMESTAMP_LEN_ERR	Length of the timestamp string exceeded the limit.
KPKG_INVALID_TIMESTAMP	The Provided time stamp is invalid. Time stamp must be in RFC 2822 format with year ranging 2000—2100.

Message	Description
KPKG_INVALID_VERSION	Version number is invalid.

Customer Keys (X509 or GPG)

GPG Key Generation

As part of securely onboarding TPA, you must generate GPG key. Use **gpg --gen-key** command to create GPG.

```
Router# gpg --gen-key
gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 1y
Key expires at Thu 21 Dec 2023 11:57:52 PM IST
Is this correct? (y/N) y
GnuPG needs to construct a user ID to identify your key.
Real name: abc
Email address: abc@cisco.com
Comment: Test GPG key for abc
You selected this USER-ID:
"abc (Test GPG key for abc) <abc@cisco.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 09E9526F marked as ultimately trusted
public and secret key created and signed.
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 3 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 3u
gpg: next trustdb check due at 2023-03-21
pub 2048R/09E9526F 2022-12-21 [expires: 2023-12-21]
Key fingerprint = DA29 846E 4B16 E0B7 7226 E57B 706C 49AE 09E9 526F
uid abc (Test GPG key for abc) <abc@cisco.com>
sub 2048R/18B50392 2022-12-21 [expires: 2023-12-21]
Router#
```

Verify the GPG Key

```
Router# gpg --list-secret-keys --keyid-format LONG
/root/.gnupg/secring.gpg
------------------------
sec 2048R/F255F66A8515763D 2022-12-16 [expires: 2023-12-16]
uid Chandan (Test GPG key) <cmohapat@cisco.com>
ssb 2048R/A181220B2E2D3898 2022-12-16
sec 2048R/B093C8FC89A0AB15 2022-12-21 [expires: 2023-03-21]
uid cmohapat (gpg key for testing purpose) <cmohapat@cisco.com>
ssb 2048R/BA8DDCD73D0958A4 2022-12-21
sec 2048R/706C49AE09E9526F 2022-12-21 [expires: 2023-12-21]
uid abc (Test GPG key for abc) <abc@cisco.com>
ssb 2048R/481345F518B50392 2022-12-21
Router#
```

Key Package Configuration File

KeyPackage Configuration File

The key package configuration file defines what operation should be done with the keys present in the key package.

Note The rules mentioned in the configuration file apply to all keys present in the key package. If you need a combination of keys, such as few keys to be added and other keys to be removed, then you must create multiple key packages—one key package to add keys, another key-package to remove keys and so on. You can then bundle these key packages into a super key package.

The key configuration file is generated when you run the Key package script on Github. The configuration file has the following fields:

Table 8: Fields in the Key Package Configuration Files

Flag	Possible Values	Mandatory Field	Purpose
VERSION	1	NO	Currently supported version.
OPERATION	ADD	YES	Creates the keys.
	DELETE		Deletes the existing keys.
TARGET	ALLOWED_ LIST	YES	Adds the keys to ALLOWED_ LIST or REVOKED_L IST.
	REVOKED_L IST		Deletes keys from allowed-list or revoked list.
USAGE	CUS-CT	YES	Application specific usage flags. The maximum length is six characters.

Flag	Possible Values	Mandatory Field	Purpose

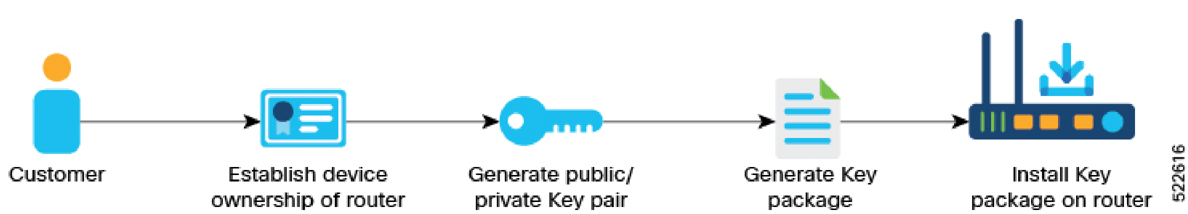
USAGE_ADDITIONAL_ DATA	<key>:<valu e>,	NO	Provides additional information related to the key such as product name, key name, and so on.
			The key and value should be separated by a colon ":" and should be delimited by comma ",".
			Example
			`USAGE_ADDITIONAL_DATA=PNM:MY_TEST_PRO DUCT_NAME`
			`Tha maximum length of this parameter is 128 characters.`

TIMESTAMP	Timestamp in RFC2822 format format	YES	To prevent replay attacks, a key package is one-time use only, which is as per the timestamp available in the key package. RFC2822 format timestamp can be generated by the command **date -R** on Linux devices.
KEYTYPE	X509KEY GPGKEY	NO	Defines the type of key being carried in the key package, either X509 or GPG key.
MULTIPLE_KEYPACKA GE	1 0	NO	This flag indicates if the given key package is a bundle or not. A bundle can contain one or more key packages. **Note** If a BUNDLE flag is set, bundle-specific configuration flags are added.
PACKAGE_LIST	ARRAY/ LIST	YES when MULTIPLE_KEYPAC KAGE is set.	A list of key package names of ALLOWED_ LISTand REVOKED_LIST keys sorted based on the timestamps with which those individual key packages are generated.

If any of the mandatory fields is missing, installation of a key package shows an error with appropriate error messages.

Onboard Key Package on Router

Figure 9: Workflow for Installing Key Package on Router



To onboard a third-party key package:

1. Generate an GPG key-pair that is used to sign the third-party key package.

See step 1 of Provisioning Key Packages on the Router, on page 53.

Note Generate your own public-private key-pair (typically this key pair is a GPG key, but it could also be an X509 certificate). This key pair is used to subsequently sign all customer software, such as RPMs.

2. Install or onboard the key pair on the Cisco IOS XR router.

   See step 2 of Provisioning Key Packages on the Router, on page 53.

Provisioning Key Packages on the Router

Before you begin

Ensure that your device ownership is established.

Procedure

Step 1 On a Linux machine, use the standard openssl commands to generate the RSA key pair.

Step 2 Generate the key package by using the script at Key Package.

Create a key package using the *create_kpkg.py tool* on Key Package.

```
create_kpkg.py -p ./oc-single.pem -r ./oc-single-priv.key  -o ADD -t ALLOWED_LIST -u KEY_ADD -i
./key_add.crt -f ./key_add.kpkg
Key package generated at: ./key_add.kpkg
```

In the following example, a key package *key_add.kpkg* is created:

```
bash-4.2$ python3 create_kpkg.py -o ADD -t ALLOWED_LIST -u "CUSTOMER-CONSENT-TOKEN" -a
"PNM:APNAM,KNM:AKNAM," -k X509KEY -i cust-ct.der -p oc-single.pem -r oc-single-priv.key -f
./key_add.kpkg
Key package generated at: ./key_add.kpkg
```

The key package is located at same directory from where you executed the above command.

Verify the generated key package by running the **verify_kpkg.py** command.

```
bash-4.2$ python3 verify_kpkg.py -p oc-single.pem -f key_add.kpkg

            Key package is valid
```

Createa key package using the *bundle_kpkg*tool

```
bundle_kpkg.py [-h] [-n] [-v] [-x TEMPDIR] -l LIST [LIST ...] -p PUBKEY -r PRIVKEY -f KEYPACKAGE
```

While creating the key package bundle, the input list of all individual key packages are sorted basedon the timestamps at which they had been created. Sorting is done off-box to reduce on- box processing. Once sorting is done, the key/file-name is arranged in sorted order in two lists which is ALLOWED_LIST and REVOKED_LIST.

On the router when a bundle is installed, first, its revoked list keys are installed in the order they are generated followed by all ALLOWED_LIST keys.

```
bundle_kpkg.py -p ./oc-single.pem -r ./oc-single-priv.key -f ./out_bundle.kpkg -l key1.kpkg key2.kpkg
 key3.kpkg key4.kpkg
```

Step 3 On the Cisco router, install the key package:

Copy the key package to the router and use the **platform security key-package customer [keypackage-bundle]** *key-package-file location* command to install the key package.

```
Router# platform security key-package customer disk0:/testing2/key-pkg/key_add.kpkg
Mon Jun 14 16:09:28.238 UTC

Key package successfully validated
Config file successfully parsed.
Successfully added key cust-ct.der to TPM
Successfully processed all keys.
Router#
```

Step 4 Verify that the key package is installed.

```
Router# show platform security key-package customer allowed-list location 0/RP0/CPU0

Mon Jun 14 16:10:01.440 UTC
---------------------------------------------
Node - node0_RP0_CPU0
---------------------------------------------

Key Name: D3CUS-CT1
Key:
MIIC7TCCAdUCAQIwDQYJKoZIhvcNAQELBQAwOzELMAkGA1UEBhMCVVMxDDAKBgNV
BAoMA3h6eTEMMAoGA1UECwwDYWJjMRAwDgYDVQQDDAdST09ULUNOMB4XDTIxMDYx
NDE1MjkwOVoXDTI0MDMxMDE1MjkwOVowPjELMAkGA1UEBhMCVVMxDDAKBgNVBAoM
A3h5ejEMMAoGA1UECwwDYWJjMRMwEQYDVQQDDApDVVNULUNULUNOMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyOT2SGTuJcQlAHCsQn4gcoZGK+po1A6g
LPV5AzOBcY0pfXV5eXoxf6S8qbmQP4l4v5MjsHzFTOuouMmijpGYFJv7TORwJ2Xw
weJ5aKbqsYTQlSQSUZ1XxG7AOdHMshVRzy7vIA7LLQJnD0j1F1U2FoRi5NhhY12L
wmYA4aPj1o+LoubAfjF1BVl3vE8rfI0mzsXODJIks+oeJbsq4HmyMbOAzLVdeucp
7bu3S8kDlc1ph4zqm81BkDZgV1++2CoCBWROt9dRZrp+ENw1GEHcXgS659iZpUmj
juG1n0W3Y6br8SE+EqqhMqkAfSbO8vaG02qYtTUNJ5gkMcTljCfDAQIDAQABMA0G
CSqGSIb3DQEBCwUAA4IBAQCDeJ5ov2gG3rj5ttpfibxiakpzl706W9crjIePJka6
CWS7Y3nxt02+PGsBByEcBPV7aU8oH2GfKN4jNZHDChfzGN7rtfRE2CG+ttvTxJLC
Ba+LjzKFSveKgPRG/gAAkZY0hRmTe7FkgmKB4UCi+u0XP3U5VlT5XRP3LGVoX0fC
rY4/GBKkG5eOF+VGD4iyPfOHjrwduO/K2DqDXyUfa1PXZDzatpnin07ShkCJQoT+
u6C1SotJ8mtrFJpePDUsa5W3O2oPROFHd4sGCivt40AbpaWECK+KLpKC+DoqN+46
tMV79rpQ0mtXo/XfY4UGir4weH9g/e2fct4g+Y2E/BD+


Key Name: D3CUS-CTX
Key:
PNM:APNAM,KNM:AKNAM,
RP/0/RP0/CPU0:router#
```

```
Router# show platform security key-package all location 0/RP0/CPU0

Mon Jun 14 16:10:01.440 UTC
---------------------------------------------
Node - node0_RP0_CPU0
---------------------------------------------

Key Name: D3CUS-CT1
Key:
MIIC7TCCAdUCAQIwDQYJKoZIhvcNAQELBQAwOzELMAkGA1UEBhMCVVMxDDAKBgNV
BAoMA3h6eTEMMAoGA1UECwwDYWJjMRAwDgYDVQQDDAdST09ULUNOMB4XDTIxMDYx
NDE1MjkwOVoXDTI0MDMxMDE1MjkwOVowPjELMAkGA1UEBhMCVVMxDDAKBgNVBAoM
A3h5ejEMMAoGA1UECwwDYWJjMRMwEQYDVQQDDApDVVNULUNULUNOMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyOT2SGTuJcQlAHCsQn4gcoZGK+po1A6g
LPV5AzOBcY0pfXV5eXoxf6S8qbmQP4l4v5MjsHzFTOuouMmijpGYFJv7TORwJ2Xw
weJ5aKbqsYTQlSQSUZ1XxG7AOdHMshVRzy7vIA7LLQJnD0j1F1U2FoRi5NhhY12L
wmYA4aPj1o+LoubAfjF1BVl3vE8rfI0mzsXODJIks+oeJbsq4HmyMbOAzLVdeucp
```

```
7bu3S8kDlc1ph4zqm81BkDZgV1++2CoCBWROt9dRZrp+ENw1GEHcXgS659iZpUmj
juG1n0W3Y6br8SE+EqqhMqkAfSbO8vaG02qYtTUNJ5gkMcTljCfDAQIDAQABMA0G
CSqGSIb3DQEBCwUAA4IBAQCDeJ5ov2gG3rj5ttpfibxiakpzl706W9crjIePJka6
CWS7Y3nxt02+PGsBByEcBPV7aU8oH2GfKN4jNZHDChfzGN7rtfRE2CG+ttvTxJLC
Ba+LjzKFSveKgPRG/gAAkZY0hRmTe7FkgmKB4UCi+u0XP3U5VlT5XRP3LGVoX0fC
rY4/GBKkG5eOF+VGD4iyPfOHjrwduO/K2DqDXyUfa1PXZDzatpnin07ShkCJQoT+
u6C1SotJ8mtrFJpePDUsa5W3O2oPROFHd4sGCivt40AbpaWECK+KLpKC+DoqN+46
tMV79rpQ0mtXo/XfY4UGir4weH9g/e2fct4g+Y2E/BD+
```


```
Key Name: D3CUS-CTX
Key:
PNM:APNAM,KNM:AKNAM,
RP/0/RP0/CPU0:router#
```

Generate Signed RPM

Cisco IOS XR supports RPM signing and signature verification for Cisco IOS XR RPM packages. All RPM packages in the Cisco IOS XR GISO and upgrade images are signed to ensure cryptographic integrity and authenticity. This guarantees that the RPM packages have not been tampered and the RPM packages are from Cisco IOS XR. Cisco creates and securely maintains the private key, which is used for signing the RPM packages.

Your applications must be available as docker images.

Packaging TPA RPMs in GISO increases GISO size. Ensure that the built GISO meets platform ISO boot size for iPXE.

Starting from Release 7.10.1, Cisco IOS XR supports signature verification of third-party signed RPM packages as well. For more information on Cisco RPMs, see *Manage Automatic Dependency* chapter.

RPM build tool for TPA is available at RPM Build Tool

GISO build tool for Signed TPA RPMs is available at: GISO Build Tool

Guidelines

- TPA RPMs must not have:
  - Scripts
  - Duplicate files
  - Dependency on Cisco packages
- RPM marked as TPA, must be installed in the same RPM directory..

Unsigned RPM

Use **ls -lrt unsigned-rpm** command to check the unsigned RPMs.

```
Router# ls -lRt unsigned-rpm/

unsigned-rpm/:

total 0
```

```
drwxr-xr-x 2 root root 92 Dec 21 20:22 v2

drwxr-xr-x 2 root root 92 Dec 21 20:22 v1
unsigned-rpm/v2:

total 187600

-rw-r--r-- 1 root root 96048752 Dec 21 20:23 owner-xyz-0.1.9-7.10.1.x86_64.rpm

-rw-r--r-- 1 root root 96048381 Dec 21 20:22 owner-abc-0.1.6-7.10.1.x86_64.rpm
unsigned-rpm/v1:

total 187600

-rw-r--r-- 1 root root 96048774 Dec 21 20:22 owner-xyz-0.1.3-7.10.1.x86_64.rpm

-rw-r--r-- 1 root root 96048375 Dec 21 20:22 owner-abc-0.1.2-7.10.1.x86_64.rpm

[root@xit-pxe-01 gpg]#
```

Signing of Unsigned RPM

Use **rpm --addsign signed-rpm/v1/owner-xyz-0.1.3-7.10.1.x86_64.rpm --macros=macros** command to sign the unsigned RPM.

```
Router# rpm --addsign signed-rpm/v1/owner-xyz-0.1.3-7.10.1.x86_64.rpm --macros=macros

Enter pass phrase:

Pass phrase is good.

signed-rpm/v1/owner-xyz-0.1.3-7.10.1.x86_64.rpm:

gpg: writing to `/var/tmp/rpm-tmp.OzeSvy.sig'

gpg: RSA/SHA256 signature from: "09E9526F abc (Test GPG key for abc) <abc@xyz.com>"

gpg: writing to `/var/tmp/rpm-tmp.HeEoUS.sig'

gpg: RSA/SHA256 signature from: "09E9526F abc (Test GPG key for abc) <abc@xyz.com>"

Router#
```

Verification of Signed RPM

```
Router# rpm -Kv signed-rpm/v1/owner-xyz-0.1.3-7.10.1.x86_64.rpm

signed-rpm/v1/owner-xyz-0.1.3-7.10.1.x86_64.rpm:

Header V3 RSA/SHA256 Signature, key ID 09e9526f: OK

Header SHA1 digest: OK (35964a3275ed2d66a6533c5cd20b6054b2547221)

V3 RSA/SHA256 Signature, key ID 09e9526f: OK

MD5 digest: OK (09eed042fbb536f5e579ae88aec95332)

Router#
```

Onboard Signed RPM Package on Router

The TPA signed RPMs are part of GISO which is onboarded on the router. For more details on building GISO with signed RPMs, see Build a Golden ISO, on page 57.

Note OV/OC packaging in GISO is not supported.

Build a Golden ISO

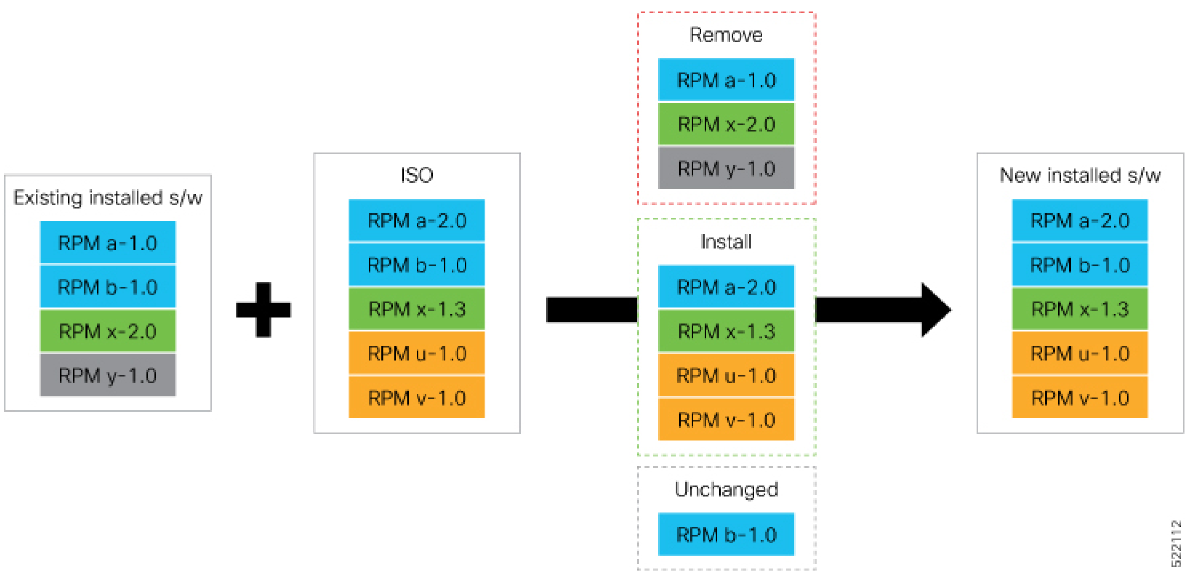
Golden ISO (GISO) upgrades the router to a version that has a predefined set of RPMs with a single operation. For example, you can create a customized ISO with the base OS package and specific optional RPMs based on your network requirements.

GISO supports automatic dependency management, and provides these functionalities:

- Builds RPM database of all the packages present in package repository.
- Skips and removes Cisco RPMs that do not match the base ISO version.
- Skips and removes third-party RPMs that are not part of already existing third-party base package in the base ISO.

For more information on building a golden ISO, see *Customize Installation using Golden ISO* chapter from *System Setup and Software Installation Guide for Cisco NCS 5500 Series Routers* guide.

Note Install operation over IPv6 is not supported.



Procedure

Step 1 Contact Cisco Support to build the GISO image with the set of packages based on your requirement.

Step 2 Build GISO image using `gisobuild.py` tool.

To build GISO, provide the following input parameters to the script:

- Base mini-x.iso (mandatory)
- Set of packages to install (Cisco signed packages)
- XR configuration file (optional)
- Label for golden ISO (optional)

Note
GISO build tool verifies the RPM dependecnies and RPM signatures. GISO build fails if the RPM is unsigned or incorrectly signed.

GISO build has the following executable requirements:

- python3 >= 3.6
- rpm >= 4.14
- cpio >= 2.10
- gzip >= 1.9
- createrepo_c
- file
- isoinfo
- mkisofs
- mksquashfs
- openssl
- unsquashfs
- 7z (Optional - but functionality may be reduced without it)

GISO build tool requires the following Python (>= 3.6) modules:

- dataclasses
- defusedxml
- distutils
- packaging
- rpm
- yaml

On a native Linux machine, th etool dependancies can be installed on supported distributions (Alma Linux 8, Fedora 34, Debian 11.2)) using `./setup/prep_dependency.sh` command.

a) Copy the repository from the Github location to an offline system or external server where the GISO will be built.

b) Run the script `gisobuild.py` and provide parameters to build the GISO image. Ensure that all RPMs and SMUs are present in the same directory or on a repository.

Example:

```
$ ./giso/src/gisobuild.py --iso <input iso> --repo <rpm repo1 rpm_repo2> \
    --pkglist <pkg1 pkg2 pkg3> --bridging-fixes <smu1 smu2 smu3> \
    --xrconfig <config.cfg> --ztp-ini <ztp.ini> --script <user_script.sh> \
    --label <label> --out-directory <out_directory> --clean./src/gisobuild.py --iso <input iso>
--repo <rpm repo1 rpm_repo2> \
    --pkglist <pkg1 pkg2 pkg3> --bridging-fixes <smu1 smu2 smu3> \
    --xrconfig <config.cfg> --ztp-ini <ztp.ini> --script <user_script.sh> \
    --label <label> --out-directory <out_directory> --clean
```

The following parameters can be provided as input to the GISO build tool:

- --iso: ISO path to mini.iso or full.iso file
- --xrconfig: XR configuration file
- --label: GISO label
- --repo: Path to repositories containing RPMs and tarballs
- --pkglist: Optional RPMs or SMUs to package
- --ztp-ini: Path to the ZTP initialization file
- --remove-packages: Remove RPMs from the GISO. To remove multiple RPMs, separate the RPM names using comma. For example, `--remove-packages xr-bgp,xr-mcast` command removes the xr-bgp and xr-mast packages from GISO
- --out-directory: Output directory to store output of the operations performed on the file
- --clean: Delete contents of the output directory
- --skip-dep-check: Skip dependency checking between files
- --version: Print version of the tool
- --pkglist: Optional RPM or SMU to package
- --yamlfile: Provide CLI arguments via YAML markup file
- --docker: Load and run pre-built docker image

The tool uses the input parameters to build the GISO image.

Use `./src/gisobuild.py --yamlfile <input_yaml_cfg>` to provide the parameters in a yaml file. To replace YAML file information, use `./src/gisobuild.py --yamlfile <input_yaml_cfg> --label <new_label>`

Step 3 Copy the GISO image to the `/harddisk:` location on the router.

Step 4 Upgrade the system to replace the current software with the .iso image, and install the RPMs.

Example:

```
Router# install replace <source location> <giso name.iso>
```

If you are using a configuration file in GISO, use the following command to extract and replace the configuration.

```
Router# install replace <source location> <GISO-with-cfg>-<platform>.iso
```

Note
The default option is to replace the existing configuration. The install operation applies the configuration from a GISO, the router reboots to activate the configuration.

Step 5 View the version information for the GISO image. You can include a label to indicate the runing software version on the router. For example, create a label v1 for the current GISO version. When you rebuild GISO with additional RPMs, you can create a label v2 to distinguish the builds.

Example:

```
Router#show version
Cisco IOS XR Software, Version  LNT
Copyright (c) 2013-2019 by Cisco Systems, Inc.

Build Information:
 Built By     : xyz
 Built On     : Sat Jun 29 22:45:27 2019
 Build Host   : iox-lnx-064
 Workspace    : ../
                //ws/
 Version      :
 Label        : -CUSTOMER_LABEL

cisco
System uptime is 41 minutes
```

```
Router#show version
Cisco IOS XR Software, Version 7.10.1 LNT
Copyright (c) 2013-2022 by Cisco Systems, Inc.
Build Information:
Built By     : xyz
Built On     : Tue June 07 19:43:44 UTC 2021
Build Host   : iox-lnx-064
Workspace    : ../ncs5500/ws
Version      : 7.10.1
Label        : 7.1.10-Customer_Label

cisco NCS5500L (D-1563N @ 2.00GHz)
cisco NCS-55A1-36H-S (D-1563N @ 2.00GHz) processor with 32GB of memory
ios uptime is 3 weeks, 1 day, 10 hours, 11 minutes
NCS-55A1-36H-S
NCS55B1 Fixed Scale HW Flexible Consumption Need Smart Lic
```

TPA Life Cycle

Application manager manages the life cycle of TPA. Following table shows the Application manager commands and their usage.

Table 9: TPA Life-Cycle Commands

Command	Description
appmgr application start name <name>	Starts an application. The application must be activated before it is started. Starting an already running application does not fail. **Example** appmgr application start name app1
appmgr application stop name <name> appmgr application kill name <name>	Stops or kills an application. The application must be activated before it can be stopped. Stopping an already stopped application does not fail. **Example** appmgr application kill name app1
appmgr application exec <name> docker-exec-cmd <cmd>	Perform a docker exec inside a given application (Docker only). **Example** appmgr application exec name app1 docker-exec-cmd ls
show appmgr application-table	Shows the basic information of an activated application in tabular format. **Example** Name Type Config State Status ---- ---- -------- ----- ---------- app1 Docker Activated Up About an hour app2 Docker Activated Exited (0) About an hour app3 Docker Activated Exited (0) About an hour app4 Docker Activated Exited (0) About an hour app5 Docker Error N/A show appmgr application name <name> info [summary|detail]
show appmgr application name <name> stats	Shows application statistics.
show appmgr application name <name> logs	Shows application logs.

Appendix

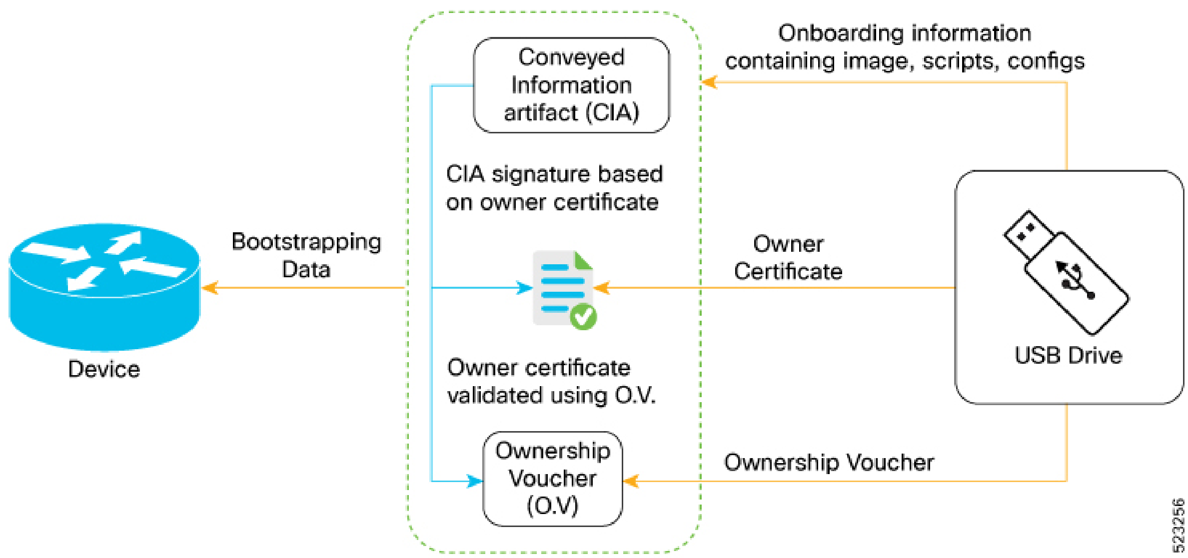
Secure ZTP Work Flow

sZTP is a technique to securely provision a networking device. Once provisioned, the device should be able to securely connect to the Network Management Systems (NMS). For more details on secure ZTP on USB, see **Secure ZTP with Removable Storage Device** section of System Setup and software Installation Guide.

Note

- Ensure no swapping of USB with another bootable USB during this reboot.
- Reimage and USB workflow does not work together.

Figure 10: Secure ZTP (USB) Work Flow



How do I build USB bootable zip file using GISO image with signed TPA RPMs?

Before you begin

- OV must be created.
- OC public key and private keys must be available.

Procedure

Step 1 Run the USB script `usb.py` and provide parameters.

Example:

```
$ ./usb.py [-h] [-prc PRECONFIG] [-c CONFIG] [-psc POSTCONFIG]
              [-ch {merge,replace}] [-iu IMAGEURL] [-ia HASHALG] [-cp] [-b -bf BOOTABLEFILE] [-ip
```

```
IMAGEDESTPATH] [-ga]
              -oc OC -ov OV -o OUTDIR -ver OSVERSION -name OSNAME -sn SERIALNUM
```

```
python3 usb.py \
        -prc scripts/pre_config_script.sh \
        -c cfg/default.cfg \
        -psc scripts/post_config_script.py \
        -ch merge \
        -iu images/NCS5500/image.iso -ia sha-256 \
        -ver 7.6.1.15I -name "Cisco IOS XR Software"  \
        -oc /auto/tftp-xrbng/akuriako/ZTP/certificates/ownercerts/pdc.cert  \
        -ocpk /auto/tftp-xrbng/akuriako/ZTP/certificates/ownercerts/pdc.key \
        -ov /auto/tftp-xrbng/akuriako/ZTP/certificates/vouchers/FOC2502R1DJ/FOC2502R1DJ.vcj \
        -sn FOC250269XE \
        -o usbdrive
```

Note

- Ensure there is no /for the image path after -iu.
- OV is created from the serial number present in the SUDI certificate.

The following parameters can be provided as input to the USB build tool:

- --prc: Pre configuration file path.
- --c: Configuration file.
- --psc: Post configuration fiel path.
- --ch: {merge,replace}: Configuration handling such as merge or replace.
- --iu: Image URL.
- --ver: Version of OS.
- --oc: Path to owner certificate.
- --ocpk: PDC key from owner certificate.
- --ov: Path to owner-ship voucher.
- --sn: RP serial number.
- --o: Output directory.

Once the OV is received, stage the USB for onboarding into device.

Step 2 Zip the file and copy to the router.

Example:

```
router# tar -zcvf usbdrive.tgz usbdrive
```

Step 3 Copy the zip file to the router.

Example:

```
Router# tar -zxvf usbdrive.tgz
```