Application Hosting Configuration Guide for Cisco NCS 5500 Series Routers, Cisco IOS XR Releases

First Published: 2015-12-23

Last Modified: 2024-06-14

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Changes to This Document

This table lists the technical changes made to this document since it was first released.

Table 1: Changes to This Document

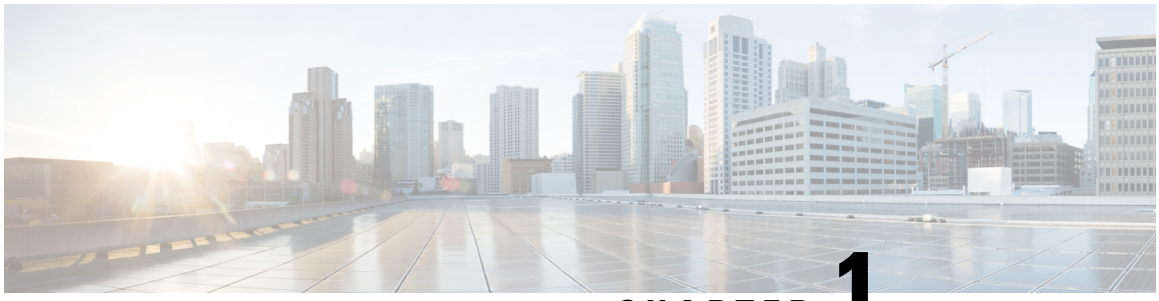
Date	Summary
June 2024	Republished for Cisco IOS XR Release 24.2.1
April 2022	Republished for Cisco IOS XR Release 7.5.2
July 2021	Republished for Cisco IOS XR Release 7.4.1
August 2019	Republished for Cisco IOS XR Release 7.0.1
March 2018	Republished for Cisco IOS XR Releases 6.4.1 and 6.3.2.
September 2017	Republished for Cisco IOS XR Release 6.3.1.
July 2017	Republished for Cisco IOS XR Release 6.2.2.
November 2016	Republished for Cisco IOS XR Release 6.1.2.
December 2015	First release for Cisco IOS XR Release 6.0.0.

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

CONTENTS



CHAPTER 1

New and Changed Feature Information

This section lists all the new and changed features for the Application Hosting Configuration Guide.

New and Changed Application Hosting Features

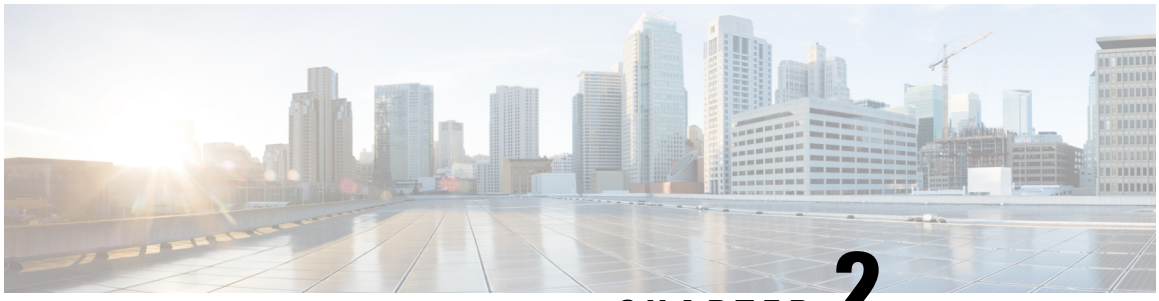
This section describes the new and changed application hosting features for Cisco IOS XR.

Application Hosting Features Added or Modified

Table 2: New and Changed Features

Feature	Description	Introduced/Changed in Release	Where Documented
CPU-Based Packet Generator	This feature was introduced.	Release 24.2.1	CPU-Based Packet Generator
Customize Docker Run Options using Application Manager	This feature was introduced.	Release 24.1.1	Customize Docker Run Options using Application Manager
Virtual IP address in the Linux networking stack	This feature was introduced.	Release 7.5.2	Communication Outside Cisco IOS XR, on page 49
On-Demand Docker Daemon Service for Hosting Applications	This feature was introduced.	Release 7.5.1	Hosting and Seamless Activation of Third Party Applications Using Application Manager, on page 63
Hosting and Seamless Activation of Third Party Applications Using Application Manager	This feature was introduced.	Release 7.3.2	Hosting and Seamless Activation of Third Party Applications Using Application Manager, on page 63

Feature	Description	Introduced/Changed in Release	Where Documented
No new features were added.	None	Release 7.4.1	NA
No new features were added.	None	Release 7.1.1	NA
No new features were added.	None	Release 7.0.1	NA
No new features were added.	None	Release 6.6.25	NA
No new features were added.	None	Release 6.5.2	NA



CHAPTER 2

Getting Started with Application Hosting

This section introduces application hosting and the Linux environment used for hosting applications on the Cisco IOS XR Operating System.

Cisco NCS 540 routers supports docker-based application hosting only.

Need for Application Hosting

Over the last decade, there has been a need for a network operating system that supports operational agility and efficiency through seamless integration with existing tool chains. Service providers have been looking for shorter product cycles, agile workflows, and modular software delivery; all of these can be automated efficiently. The 64-bit Cisco IOS XR that replaces the older 32-bit QNX version meets these requirements. It does that by providing an environment that simplifies the integration of applications, configuration management tools, and industry-standard zero touch provisioning mechanisms. The 64-bit IOS XR matches the DevOps style workflows for service providers, and it has an open internal data storage system that can be used to automate the configuration and operation of the device hosting an application.

While we are rapidly moving to virtual environments, there is an increasing need to build applications that are reusable, portable, and scalable. Application hosting gives administrators a platform for leveraging their own tools and utilities. Cisco NCS 540 routers support third-party off-the-shelf applications An application hosted on a network device can serve a variety of purposes. This ranges from automation, configuration management monitoring, and integration with existing tool chains.

Before an application can be hosted on a device, the following requirements must be met:

- Suitable build environment to build your application
- A mechanism to interact with the device and the network outside the device

When network devices are managed by configuration management applications, such as Chef and Puppet, network administrators are freed of the task of focusing only on the CLI. Because of the abstraction provided by the application, while the application does its job, administrators can now focus on the design, and other higher level tasks.

Deep Dive Into Application Hosting

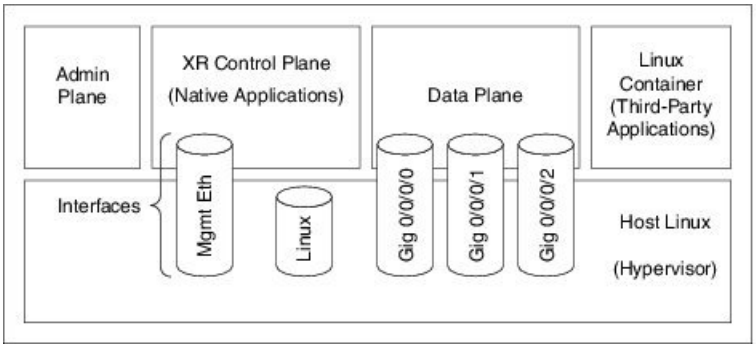
This section describes the architecture of the 64-bit IOS XR and the architecture used for application hosting.

64-bit IOS XR Architecture

IOS XR provides Linux containers for application hosting through a hypervisor. Each container provides a unique functionality. The 64-bit host Linux (hypervisor) works well with embedded systems. The various containers that are offered on the host Linux, are explained in this section.

The following figure illustrates the 64-bit IOS XR architecture.

Figure 1: 64-bit IOS XR Architecture



- **Admin Plane**: The admin plane is the first Linux container to be launched on booting IOS XR. The admin plane is responsible for managing the life cycle of the IOS XR control plane container.
- **XR Control Plane**: Applications are hosted natively in the 64-bit IOS XR control plane. You can access the IOS XR Linux bash shell through the control plane.
- **Data Plane**: The data plane substitutes and provides all the features of a line card in a modular router chassis.
- **Third-Party Container**: You can create your own Linux container (LXC) for hosting third-party applications and use the LC interfaces that are provided.

Apart from the Linux containers, several interfaces are offered on the host Linux.

Application Hosting Architecture

The 64-bit IOS XR introduces the concept of using containers on the 64-bit host Linux (hypervisor) for hosting applications in the XR control plane LXC (native) and in the third-party LXC. The host Linux is based on the Windriver Linux 7 distribution.

The application hosting architecture is designed to offer the following containers for both native and third-party applications:

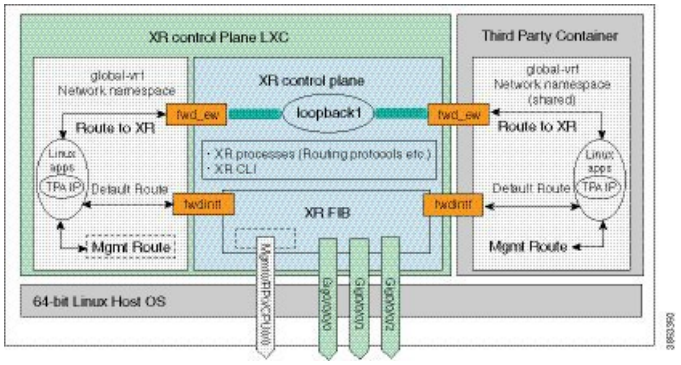
- **XR Control Plane LXC (native applications reside here)**: The XR control plane LXC contains the `global-vrf` network namespace and the XR control plane. The LXC provides the XR Linux shell to access `global-vrf` and the XR router console (CLI) to access the XR control plane. The LXC is also based on the WRL7 distribution. For more information on the XR control plane LXC.

- **Third-Party Container (third-party applications reside here)**: The 64-bit IOS XR provides you an option to create and launch your own Linux container, known as the third-party container. You can install applications within the container that shares the network namespace with XR. You can access the namespace through the XR Linux shell.

The network namespace on XR is shared across all applications and is known as `global-vrf`.

The Third-Party Application (TPA) IP is configured so that applications can communicate outside XR through the `fwdintf` interface, which is bound to the Loopback0 interface of XR. All applications communicate with XR through the `fwd_ew` interface, which is bound to the Loopback1 interface of XR.

Figure 2: Application Hosting Architecture



Note The `fwdintf` and `fwd_ew` interfaces is not support from IOS XR software release 7.9.1.

Application Hosting on the Cisco IOS XR Linux Shell

Linux supports an entire ecosystem of applications and tools that have been created, tested, and deployed by system administrators, developers, and network engineers over the last few decades. Linux is well suited for hosting servers with or without applications, because of its stability, security, scalability, reduced cost for licensing, and the flexibility it offers to customize applications for specific infrastructure needs.

With a growing focus on DevOps style workflows that focus on automation and ease of integration, network devices need to evolve and support standard tools and applications that make the automation process easier. A standardized and shared tool chain can boost speed, efficiency, and collaboration. IOS XR is developed from a Yocto-based Wind River Linux 7 distribution. The OS is RPM based and well suited for embedded systems.

IOS XR enables hosting of 64-bit Linux applications on the box, and has the following advantages:

- Seamless integration with configuration management applications
- Easy access to file systems
- Ease of operation

To host a Linux application on IOS XR, you must be familiar with the Linux shell on XR.

A typical Linux OS provides a single set of network interfaces and routing table entries that are shared across the OS. With the introduction of network namespaces, Linux provides multiple instances of network interfaces and routing tables that operate independently.

Note Support for network namespaces varies across different distributions of the Linux OS. Ensure that the distribution you are planning to use for application hosting supports network namespaces.

Network Namespaces on IOS XR

There are two ways of accessing the IOS XR Linux shell, depending on the version of Cisco IOS XR that you are using in your network.

- If you are using **Cisco IOS XR Version 6.0.0**, then you must use the procedure in Accessing the Third-Party Network Namespace on Cisco IOS XR Linux Shell, on page 6. Accessing the XR Linux shell takes you to the default network namespace, XRNNS. You must navigate from this namespace to access the third-party network namespace (TPNNS), where all the third-party application interfaces reside. There is a difference between what you can access and view at the XR router prompt, and what you can access and view at the XR Linux Shell.
- If you are using **Cisco IOS XR Version 6.0.2** and higher, then you must use the procedure in Accessing Global VRF on the Cisco IOS XR Linux Shell, on page 12. Accessing the XR Linux shell takes you directly to the third-party network namespace, renamed as global VRF. You can run bash commands at the XR router prompt itself to view the interfaces and IP addresses stored in global VRF. Navigation is faster and more intuitive in this version of IOS XR.

Accessing the Third-Party Network Namespace on Cisco IOS XR Linux Shell

The Cisco IOS XR Linux shell provides a Third-Party Network Namespace (TPNNS) that provides the required isolation between third-party applications and internal XR processes, while providing the necessary access to XR interfaces for the applications. You can use the steps mentioned in this section to access the IOS XR Linux shell and navigate through the XRNNS (default XR Network Namespace) and the TPNNS.

Note This procedure is applicable only on Cisco IOS XR Versions 5.3.2 and 6.0.0. For accessing this namespace on other versions of Cisco IOS XR, see Accessing Global VRF on the Cisco IOS XR Linux Shell, on page 12.

Use these steps to navigate through the XR Linux shell.

1. From your Linux box, access the IOS XR console through SSH, and log in.

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

   You have reached the IOS XR router prompt.

2. View the ethernet interfaces on IOS XR.

```
RP/0/0/CPU0:ios# show ipv4 interface brief
...
```

```
Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1/32      Up                    Up
GigabitEthernet0/0/0/0         10.1.1.1/24     Up                    Up
...


RP/0/RP0/CPU0:ios# show interfaces gigabitEthernet 0/0/0/0
...

GigabitEthernet0/0/0/0 is up, line protocol is up
Interface state transitions: 4
Hardware is GigabitEthernet, address is 5246.e8a3.3754 (bia
5246.e8a3.3754)
Internet address is 10.1.1.1/24
MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Duplex unknown, 1000Mb/s, link type is force-up
output flow control is off, input flow control is off
loopback not set,
Last link flapped 01:03:50
ARP type ARPA, ARP timeout 04:00:00
Last input 00:38:45, output 00:38:45
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
12 packets input, 1260 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 2 broadcast packets, 0 multicast packets
0 runts, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
12 packets output, 1224 bytes, 0 total output drops
Output 1 broadcast packets, 0 multicast packets
```

The output displays the IP and MAC addresses of the `GigabitEthernet0/0/0/0` interface.

3. Enter the **run** command to launch the IOS XR Linux bash shell.

   You can also check the version of IOS XR when you are at the bash prompt.

```
RP/0/RP0/CPU0:ios# run
Wed Oct 28 18:45:56.168 IST

[xr-vm_node0_RP0_CPU0:~]$ uname -a
Linux xr-vm_node0_RP0_CPU0 3.10.19-WR7.0.0.2_standard #1 SMP Mon Jul 6
13:38:23 PDT 2015 x86_64 GNU/Linux
[xr-vm_node0_RP0_CPU0:~]$
```

Note To exit the Linux bash shell and launch the IOS XR console, enter the **exit** command:

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

4. Locate the network interfaces by running the **ifconfig** command.

```
[xr-vm_node0_RP0_CPU0:~]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:31235 (30.5 KiB)  TX bytes:20005 (19.5 KiB)

eth-vf0   Link encap:Ethernet  HWaddr 52:54:00:34:29:44
          inet addr:10.11.12.14  Bcast:10.11.12.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe34:2944/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1566 (1.5 KiB)  TX bytes:1086 (1.0 KiB)

eth-vf1   Link encap:Ethernet  HWaddr 52:54:00:ee:f7:68
          inet6 addr: fe80::5054:ff:feee:f768/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:326483 errors:0 dropped:3 overruns:0 frame:0
          TX packets:290174 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24155455 (23.0 MiB)  TX bytes:215862857 (205.8 MiB)

eth-vf1.1794 Link encap:Ethernet  HWaddr 52:54:01:5c:55:8e
          inet6 addr: fe80::5054:1ff:fe5c:558e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:728 (728.0 B)  TX bytes:1234 (1.2 KiB)

eth-vf1.3073 Link encap:Ethernet  HWaddr e2:3a:dd:0a:8c:06
          inet addr:192.0.0.4  Bcast:192.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::e03a:ddff:fe0a:8c06/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:317735 errors:0 dropped:3560 overruns:0 frame:0
          TX packets:257881 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18856325 (17.9 MiB)  TX bytes:204552163 (195.0 MiB)

eth-vf1.3074 Link encap:Ethernet  HWaddr 4e:41:50:00:10:01
          inet addr:172.0.16.1  Bcast:172.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::4c41:50ff:fe00:1001/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:8712 errors:0 dropped:0 overruns:0 frame:0
          TX packets:32267 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:723388 (706.4 KiB)  TX bytes:11308374 (10.7 MiB)

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:1635360 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1635360 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:182532711 (174.0 MiB)  TX bytes:182532711 (174.0 MiB)

tap123    Link encap:Ethernet  HWaddr c6:13:74:4b:dc:e3
          inet6 addr: fe80::c413:74ff:fe4b:dce3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:998 (998.0 B)
```

The output displays the internal interfaces (`eth0` through `eth-vf1.3074`) used by IOS XR. These interfaces exist in XR Network Namespace (XRNNS) and do not interact with the network outside IOS XR. Interfaces that interact with the network outside IOS XR are found in the Third Party Network Namespace (TPNNS).

5. Enter the TPNNS on the IOS XR bash shell.

```
[XR-vm_node0_RP0_CPU0:~]$ ip netns exec tpnns bash
```

6. View the TPNNS interfaces.

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

The interfaces displayed in the output are replicas of the IOS XR interfaces in the Linux environment. (They have the same MAC and IP addresses.)

- `Gi0_0_0_0` is the IOS XR GigabitEthernet 0/0/0/0 interface.
- `Mg0_RP0_CPU0_0` is the IOS XR management interface, used for administrative operations on XR.

- `fwd_ew` is the interface used for communication (east to west) between third-party applications and IOS XR.
- `fwdintf` is the interface used for communication between third-party applications and the network outside IOS XR.
- `lo:0` is the IOS XR loopback0 interface used for communication between third-party applications and the outside network through the `fwdintf` interface. The loopback0 interface must be configured for applications to communicate outside XR. Alternatively, applications can also configure a GigE interface for external communication, as explained in the Communication Outside Cisco IOS XR, on page 49 section.

All interfaces that are enabled (with the **no shut** command) are added to TPNNS on IOS XR.

7. (Optional) View the IP routes used by the `fwd_ew` and `fwdintf` interfaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 1.1.1.1
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.213
```

Note The `fwdintf` and `fwd_ew` interfaces is not support from IOS XR software release 7.9.1.

Alternative Method of Entering the Third Party Network Namespace on IOS XR

To directly enter the TPNNS on logging to IOS XR, without entering the **ip netns exec tpnns bash** command, you can use the `sshd_tpnns` service, as explained in the steps that follow. The procedure involves the creation of a non-root user in order to access the service. (Root users cannot access this service.)

Note On IOS XR, prior to starting a service that binds to an interface, ensure that the interface is configured, up, and operational.

To ensure that a service starts only after an interface is configured, include the following function in the service script:

```
. /etc/init.d/tpnns-functions
tpnns_wait_until_ready
```

The addition of the **tpnns_wait_until_ready** function ensures that the service script waits for one or more interfaces to be configured before starting the service.

1. (Optional) If you want the TPNNS service to start automatically on reload, add the `sshd_tpnns` service and verify its presence.

```
bash-4.3# chkconfig --add sshd_tpnns
bash-4.3# chkconfig --list sshd_tpnns
sshd_tpnns      0:off   1:off   2:off   3:on    4:on    5:on    6:off
bash-4.3#
```

2. Start the `sshd_tpnns` service.

```
bash-4.3# service sshd_tpnns start
Generating SSH1 RSA host key: [  OK  ]
Generating SSH2 RSA host key: [  OK  ]
```

```
Generating SSH2 DSA host key: [  OK  ]
  generating ssh ECDSA key...
Starting sshd: [  OK  ]

bash-4.3# service sshd_tpnns status
sshd (pid  6224) is running...
```

3. Log into the `sshd_tpnns` session as the non-root user created in Step 1.

```
host@fe-ucs36:~$ ssh devops@192.168.122.222 -p 57722
devops@192.168.122.222's password:
Last login: Tue Sep  8 20:14:11 2015 from 192.168.122.1
XR-vm_node0_RP0_CPU0:~$
```

4. Verify whether you are in TPNNS by viewing the interfaces.

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

You are ready to use the IOS XR Linux shell for hosting applications.

Accessing Global VRF on the Cisco IOS XR Linux Shell

The Third-Party Network Namespace (TPNNS) is renamed as Global VRF (global-vrf) in Cisco IOS XR Version 6.0.2 and higher. When you access the Cisco IOS XR Linux shell, you directly enter global VRF. This is described in the following procedure.

1. From your Linux box, access the IOS XR console through SSH, and log in.

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

   You have reached the IOS XR router prompt.

2. View the ethernet interfaces on IOS XR.

```
RP/0/0/CPU0:ios# show ipv4 interface brief
...

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1/32      Up                    Up
GigabitEthernet0/0/0/0         10.1.1.1/24     Up                    Up
...


RP/0/RP0/CPU0:ios# show interfaces gigabitEthernet 0/0/0/0
...

GigabitEthernet0/0/0/0 is up, line protocol is up
Interface state transitions: 4
Hardware is GigabitEthernet, address is 5246.e8a3.3754 (bia
5246.e8a3.3754)
Internet address is 10.1.1.1/24
MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Duplex unknown, 1000Mb/s, link type is force-up
output flow control is off, input flow control is off
loopback not set,
Last link flapped 01:03:50
ARP type ARPA, ARP timeout 04:00:00
Last input 00:38:45, output 00:38:45
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
12 packets input, 1260 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 2 broadcast packets, 0 multicast packets
0 runts, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
12 packets output, 1224 bytes, 0 total output drops
Output 1 broadcast packets, 0 multicast packets
```

   The output displays the IP and MAC addresses of the `GigabitEthernet0/0/0/0` interface.

3. Verify whether the bash command runs in global VRF by running the **bash -c ifconfig** command to view the network interfaces.

```
RP/0/RP0/CPU0:ios# bash -c ifconfig
...
```

```
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

The presence of the following two interfaces confirms that you are in Global VRF:

`fwd_ew` is the interface used for communication (east to west) between third-party applications and IOS XR.

`fwdintf` is the interface used for communication between third-party applications and the network outside IOS XR.

4. Access the Linux shell by running the **bash** command.

```
RP/0/RP0/CPU0:ios# bash
Tue Aug 02 13:44:07.627 UTC
[xr-vm_node0_RP0_CPU0:~]$
```

5. (Optional) View the IP routes used by the `fwd_ew` and `fwdintf` interfaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 1.1.1.1
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.213
```

Alternative Method of Entering Global VRF on IOS XR

To directly enter global VRF on logging to IOS XR, without entering the **bash** command, you can use the `sshd_operns` service, as explained in the steps that follow. The procedure involves the creation of a non-root user in order to access the service. (Root users cannot access this service.)

Note On IOS XR, prior to starting a service that binds to an interface, ensure that the interface is configured, up, and operational.

To ensure that a service starts only after an interface is configured, include the following function in the service script:

```
. /etc/init.d/operns-functions
operns_wait_until_ready
```

The addition of the **operns_wait_until_ready** function ensures that the service script waits for one or more interfaces to be configured before starting the service.

1. (Optional) If you want the operns service to start automatically on reload, add the `sshd_operns` service and verify its presence.

```
bash-4.3# chkconfig --add sshd_operns
bash-4.3# chkconfig --list sshd_operns
sshd_operns      0:off   1:off   2:off   3:on    4:on    5:on    6:off
bash-4.3#
```

2. Start the `sshd_operns` service.

```
bash-4.3# service sshd_operns start
Generating SSH1 RSA host key: [  OK  ]
Generating SSH2 RSA host key: [  OK  ]
Generating SSH2 DSA host key: [  OK  ]
  generating ssh ECDSA key...
Starting sshd: [  OK  ]

bash-4.3# service sshd_operns status
sshd (pid  6224) is running...
```

3. Log into the `sshd_operns` session as the non-root user created in Step 1.

```
host@fe-ucs36:~$ ssh devops@192.168.122.222 -p 57722
devops@192.168.122.222's password:
Last login: Tue Sep  8 20:14:11 2015 from 192.168.122.1
XR-vm_node0_RP0_CPU0:~$
```

4. Verify whether you are in global VRF by viewing the network interfaces.

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

You are ready to use the IOS XR Linux shell for hosting applications.

Getting Started with Using Vagrant for Application Hosting

You can use vagrant as a tool for design, development, and testing of applications that can be hosted on Cisco IOS XR. You can use vagrant on a host device of your choice, for completing the steps described in the following sections.

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.

Accessing Global VRF on the Cisco IOS XR Linux Shell by Using a Vagrant Box

The Third-Party Network Namespace (TPNNS) is renamed as Global VRF (global-vrf) in Cisco IOS XR Version 6.0.2 and higher. From Cisco IOS XR Version 6.1.1 and higher, you can use a Linux-based vagrant box to directly access the Global VRF on IOS XR, as described in the following procedure.

Procedure

To access Global VRF by using a vagrant box, use the following steps.

1. Generate an API key and a CCO ID by using the steps described in https://xrdocs.github.io/getting-started/steps-download-iosxr-vagrant.
2. Download the latest stable version of the IOS XR 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. Launch the vagrant instance on your device.

```
ANNSEQUE-WS02 MINGW64:iosxrv annseque$
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'IOS-XRv'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: annseque_default_1472028191221_94197
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 57722 (guest) => 2222 (host) (adapter 1)
    default: 22 (guest) => 2223 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
```

```
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Remote connection disconnect. Retrying...
   ...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: No guest additions were detected on the base box for this VM! Guest
    default: additions are required for forwarded ports, shared folders, host only
    default: networking, and more. If SSH fails on this machine, please install
    default: the guest additions and repackage the box to continue.
    default:
    default: This is not an error message; everything may continue to work properly,
    default: in which case you may ignore this message.
==> default: Running provisioner: shell...
    default: Running: inline script
==> default: Running provisioner: shell...
    default: Running: inline script
==> default: Running provisioner: shell...
    default: Running: inline script

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

7. Access the XR Linux shell by using SSH on vagrant.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ vagrant ssh
xr-vm_node0_RP0_CPU0:~$
```

You have successfully accessed the IOS XR Linux shell.

8. (Optional) You can check the version of Linux.

```
xr-vm_node0_RP0_CPU0:~$ uname -a
Linux xr-vm_node0_RP0_CPU0 3.14.23-WR7.0.0.2_standard
#1 SMP Tue May 24 22:48:36 PDT 2016 x86_64 x86_64 x86_64 GNU/Linux
```

9. (Optional) You can view the list of available namespaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip netns list
tpnns
xrnns
global-vrf
```

10. View the network interfaces in the global VRF namespace.

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

The interfaces displayed in the output are replicas of the IOS XR interfaces in the Linux environment. (They have the same MAC and IP addresses.)

- `Gi0_0_0_0` is the IOS XR GigabitEthernet 0/0/0/0 interface.

- `Mg0_RP0_CPU0_0` is the IOS XR management interface, used for administrative operations on XR.
- `fwd_ew` is the interface used for communication (east to west) between third-party applications and IOS XR.
- `fwdintf` is the interface used for communication between third-party applications and the network outside IOS XR.
- `lo:0` is the IOS XR loopback0 interface used for communication between third-party applications and the outside network through the `fwdintf` interface. The loopback0 interface must be configured for applications to communicate outside XR. Alternatively, applications can also configure a GigE interface for external communication, as explained in the Communication Outside Cisco IOS XR, on page 49 section.

  The presence of `fwd_ew` and `fwdintf` interfaces confirm that you are in the global VRF namespace. All interfaces that are enabled (with the **no shut** command) are added to `global-vrf` on IOS XR.

11. (Optional) View the IP addresses used by the `fwd_ew` and `fwdintf` interfaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 1.1.1.1
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.213
```

12. To access the IOS XR router prompt, use the following steps.

a. Log out of the XR Linux shell virtual box.

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

b. Check the port number for accessing XR through SSH.

```
annseque@ANNSEQUE-WS02 MINGW64 ~
$ vagrant port
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)
```

c. Use the port number, **2223**, and the password, **vagrant**, for accessing XR through SSH .

```
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 have successfully accessed the XR router prompt.

13. View the network interfaces by using the **bash -c ifconfig** command at the XR router prompt.

```
RP/0/RP0/CPU0:ios# bash -c ifconfig
Thu Jul 21 06:03:49.098 UTC

Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

You can view all the interfaces available in global VRF namespace through the XR router prompt.

14. (Optional) To navigate to the XR Linux shell, you can use the **run** command. To navigate back to the router prompt, you can use the **exit** command.

```
RP/0/RP0/CPU0:ios# run
Thu Jul 21 05:57:04.232 UTC

[xr-vm_node0_RP0_CPU0:~]$

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

You are ready to use the IOS XR Linux shell for hosting applications.

Applying Bootstrap Configuration to Cisco IOS XR by Using a Vagrant Box

Configuration that is applied to a router or a device during boot-up is known as bootstrap configuration. By using a vagrant box, you can create a bootstrap configuration and apply it to an instance of the Cisco IOS XR running on a vagrant box.

Procedure

To bootstrap configuration to an instance of XR running on a vagrant box, use the following steps.

1. Generate an API key and a CCO ID by using the steps described in https://xrdocs.github.io/getting-started/steps-download-iosxr-vagrant.
2. Download the latest stable version of the IOS XR 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 vagrant file containing the configuration with which you want to bootstrap the XR.

```
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 single_node_bootstrap/
```

```
configs/  scripts/  Vagrantfile
```

8. Create the bootstrap configuration file which uses a vagrant shell provisioner.

   You would need a shell provisioner section for each node in your network. A sample configuration file is as follows:

```
#Source a config file and apply it to XR

 config.vm.provision "file", source: "configs/rtr_config", destination:
"/home/vagrant/rtr_config"

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

   In the shown sample file, you are using a vagrant file provisioner (`config.vm.provision "file"`) to transfer a file from your host machine to the XR Linux shell. The root of the source directory is the working directory for your vagrant instance. Hence, the `rtr_config` file is located in the `configs` directory.

   You are using a shell script (`config.vm.provision "shell"`) to apply the bootstrap configuration to XR. The shell script eventually runs on the XR Linux shell of the vagrant instance. This script is placed in the `scripts` directory and is named as `apply_config.sh`. The script uses the location of the router configuration file as the destination parameter in the vagrant file provisioner.

9. Verify the directory structure for the single node bootstrap configuration example used in this section.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd single_node_bootstrap/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_bootstrap (master)
$ tree ./
./
├── Vagrantfile
├── configs
│   └── rtr_config
└── scripts
    └── apply_config.sh

2 directories, 3 files
```

10. Verify the contents of the bootstrap configuration file.

    The bootstrap configuration example we are using in this section configures the gRPC server on port 57789.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_bootstrap (master)
$ cat configs/rtr_config
!! XR configuration
!
grpc
  port 57789
!
end
```

Note The bootstrap configuration is appended to the existing configuration on the instance of XR.

11. Verify the contents of the shell script you are using to apply the configuration to XR.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_bootstrap (master)
$ cat scripts/apply_config.sh
#!/bin/bash

## Source ztp_helper.sh to get the xrapply and xrcmd functions.
source /pkg/bin/ztp_helper.sh

function configure_xr()
{
   ## Apply a blind config
   xrapply $1
   if [ $? -ne 0 ]; then
       echo "xrapply failed to run"
   fi
   xrcmd "show config failed" > /home/vagrant/config_failed_check
}

## The location of the config file is an argument to the script
config_file=$1

## Call the configure_xr() function to use xrapply and xrcmd in parallel
configure_xr $config_file

## Check if there was an error during config application
grep -q "ERROR" /home/vagrant/config_failed_check

## Condition based on the result of grep ($?)
if [ $? -ne 0 ]; then
    echo "Configuration was successful!"
    echo "Last applied configuration was:"
    xrcmd "show configuration commit changes last 1"
else
    echo "Configuration Failed. Check /home/vagrant/config_failed on the router for
logs"
    xrcmd "show configuration failed" > /home/vagrant/config_failed
    exit 1
fi
```

In this example, the shell script blindly applies the configuration file specified as an argument ($1) and then checks to see if there was an error while applying the configuration.

The following new commands are introduced in the shell script:

- **xrcmd**: Allows you to run privileged exec commands at the XR router prompt on the XR Linux shell.

  For example, **show run**, **show version**, and so on.

- **xrapply**: Allows you to apply (append) a configuration file to the existing configuration.

- **xrapply_string**: Applies a configuration directly using a single inline string.

  For example, **xrapply_string "interface Gig0/0/0/0\n ip address 1.1.1.2/24 \n no shutdown**

Note To enable the **xrapply**, **xrapply_string**, and **xrcmd** commands`source /pkg/bin/ztp_helper.sh`, it is mandatory to include `source /pkg/bin/ztp_helper.sh` in the script.

12. Verify if the shell provisioner code has been included in the vagrant file.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_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.box = "IOS-XRv"

    #Source a config file and apply it to XR

    config.vm.provision "file", source: "configs/rtr_config", destination:
"/home/vagrant/rtr_config"

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

13. Launch the vagrant instance from the current directory.

Launching the vagrant instance should bootstrap the configuration to XR.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_bootstrap (master)
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'IOS-XRv'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM:
single_node_bootstrap_default_1472117544017_81536
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 57722 (guest) => 2222 (host) (adapter 1)
    default: 22 (guest) => 2223 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Remote connection disconnect. Retrying...
    ...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
```

```
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: No guest additions were detected on the base box for this VM! Guest
    default: additions are required for forwarded ports, shared folders, host only
    default: networking, and more. If SSH fails on this machine, please install
    default: the guest additions and repackage the box to continue.
    default:
    default: This is not an error message; everything may continue to work properly,
    default: in which case you may ignore this message.
==> default: Running provisioner: shell...
    default: Running: inline script
==> default: Running provisioner: shell...
    default: Running: inline script
==> default: Running provisioner: shell...
    default: Running: inline script
==> default: Running provisioner: file...
==> default: Running provisioner: shell...
    default: Running:
C:/Users/annseque/AppData/Local/Temp/vagrant-shell20160825-3292-1wncpa3.sh
==> default: Configuration was successful!
==> default: Last applied configuration was:
==> default: Building configuration...
==> default: !! IOS XR Configuration version = 6.1.1.18I
==> default: grpc
==> default:  port 57789
==> default: !
==> default: end

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

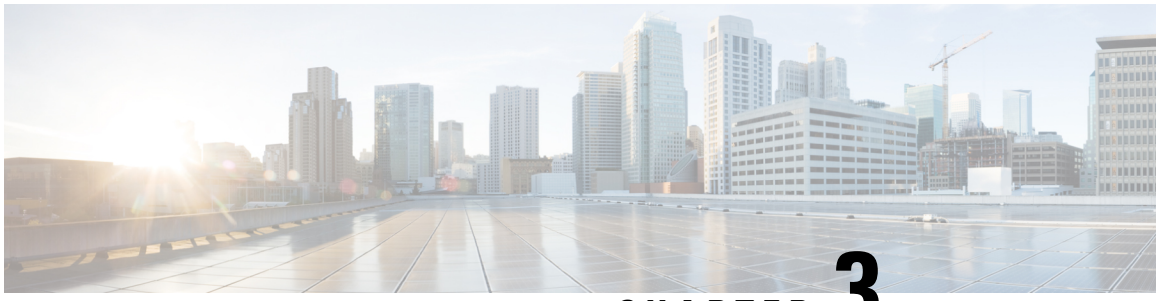
You can see the vagrant file and shell provisioner applying the gPRC server port configuration to XR.

14. (Optional) You can verify the bootstrap configuration on the XR router console from the XR Linux shell.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/single_node_bootstrap (master)
$ vagrant port
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)


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


RP/0/RP0/CPU0:ios# show running-config grpc
Thu Aug 25 09:42:24.010 UTC
grpc
 port 57789
!

RP/0/RP0/CPU0:ios# show configuration commit changes last 1
Thu Aug 25 09:42:34.971 UTC
Building configuration...
!! IOS XR Configuration version = 6.1.1.18I
grpc
 port 57789
!
end

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

You have successfully applied a bootstrap configuration to XR.



CHAPTER 3

Accessing the Networking Stack

The Cisco IOS XR Software serves as a networking stack for communication. This section explains how applications on IOS XR can communicate with internal processes, and with servers or outside devices.

Packet I/O on IOS XR

This section illustrates how, with the Packet I/O functionality, you can use Linux applications to manage communication with the IOS XR interfaces. It describes how the OS environment must be set up to establish packet I/O communication with hosted applications.

Exposed IOS-XR Interfaces in Linux

Feature Name	Release Information	Description
Automatic Synchronization of Secondary IPv4 addresses from XR to Linux OS	Release 7.9.1	Now the configured interface secondary IPv4 addresses on the Cisco IOS XR software are automatically synchronized to Linux operating system. The third-party applications on Cisco IOS XR can use the secondary IPv4 addresses without any manual intervention. Earlier, you had to configure the secondary IPv4 addresses on the Linux operating system manually.

The secondary IPv4 addresses that are configured for an XR interface are now synchronized into the Linux operating system automatically. With this secondary IPv4 address synchronization, the third party applications that are deployed on Cisco IOS XR can now use the secondary IPv4 addresses. Prior to this release, only

primary IPv4 addresses were supported and the secondary IPv4 addresses had to be configured manually in the Linux operating system.

Exposed XR interfaces (EXIs) and address-only interfaces support secondary IPv4 address synchronization:

- EXIs have secondary IP addresses added to their corresponding Linux interface
- Address-only interfaces have secondary IP addresses added to the Linux loopback device. For additional information on address-only interfaces, see show linux networking interfaces address-only.

The restrictions of secondary IPv4 addresses synchronization are:

- Secondary IPv4 addresses are not synchronized from Linux to XR for Linux-managed interfaces.
- The **ifconfig** Linux command only displays the first configured IPv4 address. To view the complete list of IPv4 addresses, use the **ip addr show** Linux command.

For additional information on secondary IPv4 addresses, see ipv4 address (network).

You can run **bash** commands at the IOS XR router prompt to view the interfaces and IP addresses stored in global VRF. When you access the Cisco IOS XR Linux shell, you directly enter the global VRF.

SUMMARY STEPS

1. From your Linux box, access the IOS XR console through SSH, and log in.
2. View the ethernet interfaces on IOS XR.
3. Check the IP and MAC addresses of the interface that is in `Up` state. Here, interfaces `HundredGigE0/0/0/24` and `MgmtEth0/RP0/CPU0/0` are in the `Up` state.
4. Verify that the bash command runs in global VRF to view the network interfaces.
5. Access the Linux shell.
6. (Optional) View the IP routes used by the `to_xr` interfaces.

DETAILED STEPS

Step 1 From your Linux box, access the IOS XR console through SSH, and log in.

Example:

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
Router#
```

Step 2 View the ethernet interfaces on IOS XR.

Example:

```
Router#show ip interface brief
Interface IP-Address Status Protocol Vrf-Name
FourHundredGigE0/0/0/0 unassigned Shutdown Down default
FourHundredGigE0/0/0/1 unassigned Shutdown Down default
FourHundredGigE0/0/0/2 unassigned Shutdown Down default
FourHundredGigE0/0/0/3 unassigned Shutdown Down default
FourHundredGigE0/0/0/4 unassigned Shutdown Down default
FourHundredGigE0/0/0/5 unassigned Shutdown Down default
FourHundredGigE0/0/0/6 unassigned Shutdown Down default
FourHundredGigE0/0/0/7 unassigned Shutdown Down default
FourHundredGigE0/0/0/8 unassigned Shutdown Down default
FourHundredGigE0/0/0/9 unassigned Shutdown Down default
```

```
FourHundredGigE0/0/0/10 unassigned Shutdown Down default
FourHundredGigE0/0/0/11 unassigned Shutdown Down default
FourHundredGigE0/0/0/12 unassigned Shutdown Down default
FourHundredGigE0/0/0/13 unassigned Shutdown Down default
FourHundredGigE0/0/0/14 unassigned Shutdown Down default
FourHundredGigE0/0/0/15 unassigned Shutdown Down default
FourHundredGigE0/0/0/16 unassigned Shutdown Down default
FourHundredGigE0/0/0/17 unassigned Shutdown Down default
FourHundredGigE0/0/0/18 unassigned Shutdown Down default
FourHundredGigE0/0/0/19 unassigned Shutdown Down default
FourHundredGigE0/0/0/20 unassigned Shutdown Down default
FourHundredGigE0/0/0/21 unassigned Shutdown Down default
FourHundredGigE0/0/0/22 unassigned Shutdown Down default
FourHundredGigE0/0/0/23 unassigned Shutdown Down default
HundredGigE0/0/0/24 10.1.1.10 Up Up default
HundredGigE0/0/0/25 unassigned Shutdown Down default
HundredGigE0/0/0/26 unassigned Shutdown Down default
HundredGigE0/0/0/27 unassigned Shutdown Down default
HundredGigE0/0/0/28 unassigned Shutdown Down default
HundredGigE0/0/0/29 unassigned Shutdown Down default
HundredGigE0/0/0/30 unassigned Shutdown Down default
HundredGigE0/0/0/31 unassigned Shutdown Down default
HundredGigE0/0/0/32 unassigned Shutdown Down default
HundredGigE0/0/0/33 unassigned Shutdown Down default
HundredGigE0/0/0/34 unassigned Shutdown Down default
HundredGigE0/0/0/35 unassigned Shutdown Down default
MgmtEth0/RP0/CPU0/0 192.168.122.22 Up Up default
```

Note Use the **ip addr show** or **ip link show** commands to view all corresponding interfaces in Linux. The IOS XR interfaces that are `admin-down` state also reflects a `Down` state in the Linux kernel.

Step 3 Check the IP and MAC addresses of the interface that is in `Up` state. Here, interfaces `HundredGigE0/0/0/24` and `MgmtEth0/RP0/CPU0/0` are in the `Up` state.

Example:

```
Router#show interfaces HundredGigE0/0/0/24
...
HundredGigE0/0/0/24 is up, line protocol is up
Interface state transitions: 4
Hardware is HundredGigE0/0/0/24, address is 5246.e8a3.3754 (bia
5246.e8a3.3754)
Internet address is 10.1.1.1/24
MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Duplex unknown, 1000Mb/s, link type is force-up
output flow control is off, input flow control is off
loopback not set,
Last link flapped 01:03:50
ARP type ARPA, ARP timeout 04:00:00
Last input 00:38:45, output 00:38:45
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
12 packets input, 1260 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 2 broadcast packets, 0 multicast packets
0 runs, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
12 packets output, 1224 bytes, 0 total output drops
Output 1 broadcast packets, 0 multicast packets
```

Step 4 Verify that the bash command runs in global VRF to view the network interfaces.

Example:

```
Router#bash -c ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:360 (360.0 B) TX bytes:0 (0.0 B)
Mg0_RP0_CPU0_0 Link encap:Ethernet HWaddr 54:00:00:00:bd:49
inet addr:192.168.122.22 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:3859 errors:0 dropped:0 overruns:0 frame:0
TX packets:1973 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2377782 (2.2 MiB) TX bytes:593602 (579.6 KiB)
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:242 errors:0 dropped:0 overruns:0 frame:0
TX packets:242 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:12100 (11.8 KiB) TX bytes:12100 (11.8 KiB)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:60 (60.0 B)
```

The `to_xr` interface indicates access to the global VRF.

Step 5 Access the Linux shell.

Example:

```
Router#bash
[ios:~]$
```

Step 6 (Optional) View the IP routes used by the `to_xr` interfaces.

Example:

```
[ios:~]$ip route
default dev to_xr scope link metric 2048
6.1.0.0/16dev Mg0_RP0_CPU0_0 proto kernel scope link src 6.1.22.41
20.1.0.0/16dev Hu0_0_0_0 proto kernel scope link src 20.1.1.1
20.2.0.0/16dev Hu0_0_0_20 proto kernel scope link src 20.2.1.1
30.1.0.0/24dev BE500 proto kernel scope link src 30.1.0.1
172.17.0.0/16dev docker0 proto kernel scope link src 172.17.0.1linkdown
```

Note You can also enter the global VRF directly after logging into IOS XR using the **run ip netns exec vrf-default bash** command.

Setting up Virtual IP Addresses

Feature Name	Release Information	Description
Virtual IP address in the Linux networking stack	Release 7.5.2	Virtual IP addresses allow a single IP address to connect to the current active RP after an RP switchover event. In addition, this functionality enables your network stack to support virtual IP addresses for third-party applications and IOS XR applications that use the Linux networking stack. The following commands are modified: • ipv4 virtual address • ipv6 virtual address • show linux networking interfaces address-only

Interfaces configured on IOS XR are programmed into the Linux kernel. These interfaces allow Linux applications to run as if they were running on a regular Linux system. This packet I/O capability ensures that off-the-shelf Linux applications can be run alongside IOS XR, allowing operators to use their existing tools and automate deployments with IOS XR.

The IP address on the Linux interfaces, MTU settings, MAC address are inherited from the corresponding settings of the IOS XR interface. Accessing the global VRF network namespace ensures that when you issue the **bash** command, the default or the global VRF in IOS XR is reflected in the kernel. This ensures default reachability based on the routing capabilities of IOS XR and the packet I/O infrastructure.

Virtual addresses can be configured to access a router from the management network such as gRPC using a single virtual IP address. On a device with two or more RPs, the virtual address refers to the management interface that is currently active. This functionality can be used across RP failover without the information of which RP is currently active. This is applicable to the Linux packet path.

Procedure

	Command or Action	Purpose
Step 1	You can use the following commands to verify the IP Address in the Linux networking stack:	• ipv4 virtual address • ipv6 virtual address • show linux networking interfaces address-only

Third-Party Application Networking in Named VRFs

Feature Name	Release Information	Description
Virtual Routing and Forwarding for Linux Third-Party Applications using Data Port	Release 7.9.1	This feature empowers you to run your native Linux applications on Cisco IOS XR as-is, without any modifications. You can now configure a host of utilities that allows for easy integration of Linux devices and applications. These utilities allow applications hosted in containers to interact with native Cisco IOS XR applications (hosted in the XR control plane). The following commands are modified:**show linux networking vrfs**.

Cisco IOS XR now supports the use of standard Linux APIs to send and receive packets, update routes, interface state, interface IP addresses, and so on.

The supported utilities are:

- Default Route Source Address
- East-West Communication
- Hardware LPTS Support for Traffic Protection
- Management Route Export
- Automatic Mapping of Deprecated TPA Configuration
- Software Forwarding
- Statistics Synchronization
- VRF Disable

Default Route Source Address

The *Default Route Source Address* utility allows you to specify an interface in which the address should be used as the *source hint* on Linux's default route.

This source hint is used for traffic where:

- The application is not bound to a specific address.
- The traffic is destined over a nonconnected route. This is commonly seen as *Rx-inject traffic* and represents most of the traffic that is sent by Linux.

Ensure that the interface is synchronized to Linux, to qualify as a valid source hint interface.

- Its VRF must not be disabled.
- On XR platforms, it must not be the East-West interface.
- It is a supported interface type.
- If explicitly configured, it must be in the specified VRF.

The following configuration parameters are used to select the interface to be used:

- If an interface is specified explicitly and valid, it is used.
- If **active-management** is specified, the lowest-numbered valid management interface on the active RP is used. The identity of this interface will change after RP switchover.
- If no configuration is specified, the lowest-numbered valid loopback interface in the VRF is used.

The address that is chosen from the selected source hint interface depends on the address family:

- IPv4: The primary address is used, when present. Secondary addresses are not considered.
- IPv6: The IP address that is numerically the lowest is used.

Following is the configuration for setting the default source hint interface address:

```
vrf blue
!
linux networking
 vrf blue
  east-west Loopback3
  address-family ipv4
   source-hint default-route interface Loopback2
  !
  address-family ipv6
   source-hint default-route interface Loopback2
  !
 !
!
interface Loopback2
 vrf blue
 ipv4 address 192.0.2.1 255.255.255.255
 ipv6 address 2001:db8::1/128
!
interface Loopback3
 vrf blue
 ipv6 address 2001:db8::ea57/128
!
```

Use the following show command to verify whether the default source hint interface address is configured:

```
RP/0/RP0/CPU0:ios#show linux networking vrfs vrf blue
VRF blue (Linux network namespace vrf-blue):
  Status:                          active
  IPv4 default route source hint: 192.0.2.1
  IPv6 default route source hint: 2001:db8::1
  IPv4 XR East-West:               none
  IPv6 XR East-West:               2001:db8::ea57
```

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
  vrf < vrf-name >
```

```
address-family { ipv4 | ipv6 }
  update-source dataports { < interface > | active-management }
```

East-West Communication

The *East-West Communication* utility allows you to specify a Cisco IOS XR interface that should be used for communication between Linux and Cisco IOS XR applications.

Configuring an interface as East-West for a virtual routing and forwarding (VRF) ensures that all listed addresses are reserved for East-West communication, with the following behaviour:

- Traffic cannot be routed from Linux to other devices using this IP address.
- Traffic destined to the listed addresses cannot be received by Linux applications.
- The IP addresses will not appear in Linux.
- For Linux applications: Traffic might be sourced from any local IP address present in Linux. Traffic must be sent to one of the reserved East-West IP addresses.
- For Cisco IOS XR applications: Traffic must be sourced from one of the reserved East-West IP addresses. Traffic might be sent to any local IP address present in Linux.

Ensure the following, for the interface to be qualified as a valid East-West interface:

- Be in a VRF that is not disabled.
- Have one or more IP addresses.
- The following configuration is used to select the interface to be used:
  - If an interface is specified explicitly and valid, it is used.
  - If no configuration is specified, Loopback1 is used.
- All IP addresses on the interface are reserved for East-West.

Following is the configuration to define the East-West communication:

```
vrf blue
!
linux networking
 vrf blue
  east-west Loopback3
  address-family ipv4
   source-hint default-route interface Loopback2
  !
  address-family ipv6
   source-hint default-route interface Loopback2
  !
 !
!
interface Loopback2
 vrf blue
 ipv4 address 192.0.2.1 255.255.255.255
 ipv6 address 2001:db8::1/128
!
interface Loopback3
 vrf blue
```

```
 ipv6 address 2001:db8::ea57/128
!
```

Use the following show command to verify whether the east-west communication is configured:

```
RP/0/RP0/CPU0:ios#show linux networking vrfs vrf blue
VRF blue (Linux network namespace vrf-blue):
  Status:                          active
  IPv4 default route source hint: 192.0.2.1
  IPv6 default route source hint: 2001:db8::1
  IPv4 XR East-West:               none
  IPv6 XR East-West:               2001:db8::ea57
```

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
  vrf < vrf-name >
    east-west < interface >
```

Hardware LPTS Support For Traffic Protection

The *Hardware Local Packet Transport Services (LPTS) Support for Traffic Protection* utility allows you to specify traffic protection rules to be factored into as an LPTS programming that is done by the Linux Packet I/O. This is in addition to the existing method where the rules were implemented using the Linux kernel's software-based *nftables firewall*. The nftables firewall is a subsystem of the Linux kernel, and provides filtering and classification of network packets. The nftables firewall is retained as a fallback, but augmented by higher performance LPTS rules.

Linux Packet I/O programs the LPTS in response to Linux socket operations, to ensure that Linux clients can receive traffic from other devices. When traffic protection rules are configured, this feature applies filtering to the programmed LPTS rules to allow a restricted subset that matches the traffic protection rules.

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
  vrf < vrf-name >
    address-family { ipv4 | ipv6 }
      protection
        allow protocol { tcp | udp } local-port < local-port >

          {  remote-address < remote-address >/< prefix-len >
           | local-address  < local-address >/< prefix-len >
           | interface      < interface-name > }
```

Management Route Export

The *Management Route Export* utility allows for a subset of Cisco IOS XR static routes that resolve over the active management interfaces to be replicated to Linux. This avoids the need for a line card NPU inject and FIB lookup for routing Linux traffic matching these management routes.

In order for the routes to be exported from Cisco IOS XR to Linux, you must ensure that the routes:

- Resolve over the management interface.
- Are static.
- Not recursive.
- Not the default XR route.

A specified source hint interface is qualified only if:

- Its VRF is not disabled.
- The interface is in the same VRF as the management interface.
- On Cisco IOS XR platforms, it is not the East-West interface.
- It is a supported interface type.

If the specified interface is valid, then the address that is chosen from it depends on the address family:

- IPv4: The primary address is used, when present. Secondary addresses are not considered.
- IPv6: The IP address that is numerically the lowest is used.

The following configuration allows for the source hint interface to be specified for the static routes which are synchronized into Linux.

```
linux networking
 vrf default
  address-family ipv4
   source-hint management-route interface Loopback2
  !
 !
!
interface Loopback0
ipv4 address 192.0.2.128 255.255.255.255
!
interface Loopback2
 ipv4 address 192.0.2.200 255.255.255.255
!
interface MgmtEth0/RP0/CPU0/0
 ipv4 address 192.0.2.1 255.255.255.240
!
router static
 address-family ipv4 unicast
  192.0.2.16/28 192.0.2.2
  192.0.2.32/28 192.0.2.2
 !
!
```

Use the following show command to verify whether the east-west communication is configured:

Note The management ethernet is directly connected to a device with the unicast route IP address.

```
RP/0/RP0/CPU0:ios#bash vrf default ip route
default dev to_xr scope link src 192.0.2.128 metric 2048 mtu 1500 advmss 1460
192.0.2.1/30 dev Mg0_RP0_CPU0_0 proto static scope link src 192.0.2.200
192.0.16.0/24 via 192.0.2.2 dev Mg0_RP0_CPU0_0 proto static src 192.0.2.200 metric 2048
192.0.17.0/24 via 192.0.2.2 dev Mg0_RP0_CPU0_0 proto static src 192.0.2.200 metric 2048
```

The verification for source hint config is to check that all Linux routes resolving via the management ethernet interface are using the source address from the configured device. The verification for management route export is to check that all static routes resolving via the management ethernet interfaceare exported to Linux.

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
```

```
vrf < vrf-name >
  address-family { ipv4 | ipv6 }
    update-source destination < management-interface > source < interface >
```

Mapping of Deprecated TPA Configuration

The *Automatic Mapping of Deprecated TPA Configuration* utility supports seamless migrations from a Cisco IOS XR environment to Linux, with the Packet I/O functionality. The configuration is translated from the deprecated TPA configuration (under **tpa**) to Linux, with the Packet I/O configuration (under **linux networking**).

The configuration will be automatically translated to the equivalent Linux Packet I/O configuration, after installation.

The following scenarios are relevant for this utility:

- Applying deprecated TPA configuration on a Cisco IOS XR device that supports Linux Packet I/O.
- Upgrading a Cisco IOS XR device from a version that does not support Linux Packet I/O, to a version that supports Linux Packet I/O.
- Downgrading a Cisco IOS XR device from a version that supports Linux Packet I/O, to a version that does not support Linux Packet I/O.
- The deprecated configuration is available until all Cisco XR platforms are migrated to support Linux Packet I/O.

Note Downgrading to an unsupported version of Linux Packet I/O cannot be done automatically. The definitions required to support Linux Packet I/O configuration does not exist on releases earlier to Cisco IOS XR Release 7.9.1.

Software Forwarding

The *Software Forwarding* utility allows you to choose software forwarding over hardware forwarding. Software forwarding is provided primarily for compatibility with Cisco IOS XR networking stack, where hardware forwarding could not route packets over the management interface.

When software forwarding is configured, the Net I/O will be used for forwarding packets. The packet path might be slow, although no change to Linux reachability is noticeable. You can use software forwarding to avoid injecting traffic toward line card NPUs in scenarios where the Linux traffic in a VRF will be sent over management interfaces.

Following is the configuration for software forwarding:

```
linux networking
 vrf default
  address-family ipv6
   default-route software-forwarding
  !
 !
!
```

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
```

```
vrf < vrf-name >
  address-family { ipv4 | ipv6 }
    default-route mgmt
```

Statistics Synchronization

The *Statistics Synchronization* utility allows you to specify the intervals when interface statistics for all interfaces are synchronized to Linux, when using the Linux **ethtool** interface, to gather interface statistics.

For supported configurations, Cisco IOS XR's **statsd infra** is polled at specified intervals to retrieve cached interface statistics for all interfaces that are exposed to Linux, as an exposed Cisco IOS XR interface (those visible to the Linux **ip link** command).

However, statistics are not gathered for interfaces in disabled VRFs, or for those interfaces which are not synchronized to Linux as an exposed interface.

This example shows how the bundle-ether interface packet statistics are synchronized between Cisco IOS XR and Linux. The packet and byte counters that are maintained by Linux for Cisco IOS XR interfaces display only the traffic that is sourced in Linux. You can configure to periodically synchronize these counters with the Cisco IOS XR statistics for the interfaces.

1. Following is the configure for statistics synchronization, including the direction and synchronization interval.

   ```
   linux networking
     statistics-synchronization from-xr every { 30s | 60s | 2m | 3m | 4m | 5m | 6m | 7m |
   8m | 9m | 10m }
   ```

   The following example shows statistics synchronization in global configuration:

   ```
   Router(config)#linux networking statistics-synchronization from-xr
   every 30s
   ```

   The following example shows statistics synchronization in exposed-interface configuration:

   ```
   Router(config)#linux networking exposed-interfaces interface
   bundle-ether 1 statistics-synchronization from-xr every 10s
   ```

   where—

   - **from-xr:** The direction indicating that the interface packet statistics will be pushed from Cisco IOS XR to the Linux kernel.
   - **every:** Shows the frequency at which to synchronize statistics. The intervals that are supported for global configuration are 30s and 60s. The intervals that are supported for exposed interfaces are 5s, 10s, 30s, or 60s. The interval `s` is in seconds.

2. Verify that the statistics synchronization is applied successfully on Cisco IOS XR.

   ```
   Router#show run linux networking
   linux networking
    vrf default
     address-family ipv4
      protection
      protocol tcp local-port all default-action deny
       permit interface bundle-ether 1
       !
      !
     !
    !
   exposed-interfaces
    interface bundle-ether 1 linux-managed
   ```

```
    statistics-synchronization from-xr every 10s
    !
   !
  !
```

You can use the **show tech-support linux networking** command to display debugging information, with regard to statistics synchronisation.

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
  statistics update-frequency < 1 - 99999999 >
```

Note The integer values here are mapped to the nearest matching value in supported configuration:

- For values not exceeding 600 seconds, it is corrected to the nearest matching interval.
- For values exceeding 600 seconds, it is corrected to 10 minutes.

VRF Disable

The *VRF Disable* utility enables you to specify the virtual routing and forwarding (VRF) that should not be synchronized to Linux, and will not be used by applications using the Linux packet path. This configuration improves performance. Communication using Linux Packet I/O (including East-West communication) will not be functional in the VRF or network namespace which was disabled.

The usage of the VRF Disable utility depends on whether you are using the Cisco IOS XR default VRF or the nondefault VRF:

- For the default VRF, no interfaces, routes, or addresses are synchronized to Linux, but a network namespace called "vrf-default" still exists.
- For nondefault VRFs, the corresponding network namespace is deleted.

You can run the VRF Disable utility by using the following configuration:

```
vrf green
!
linux networking
 vrf green
  disable
 !
!
```

Use the following show command to verify whether the VRF is disabled:

```
RP/0/RP0/CPU0:ios#show linux networking vrfs vrf green
VRF green (Linux network namespace not created):
  Status:                          VRF disabled
```

The following TPA configuration has been deprecated, from Cisco IOS XR Release 7.9.1:

```
tpa
  vrf < vrf-name >
    disable
```

Program Routes in Linux

The basic routes required to allow applications to send or receive traffic can be programmed into the kernel. The Linux network stack that is part of the kernel is used by normal Linux applications to send/receive packets. In an IOS XR stack, IOS XR acts as the network stack for the system. Therefore to allow the Linux network stack to connect into and use the IOS XR network stack, basic routes must be programmed into the Linux Kernel.

Step 1 View the routes from the bash shell.

Example:

```
[ios:~]$ip route
default dev to_xr scope link src 10.1.1.10 metric 2048
10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
```

Step 2 Programme the routes in the kernel.

Two types of routes can be programmed in the kernel:

- **Default Route:** The default route sends traffic destined to unknown subnets out of the kernel using a special `to_xr` interface. This interface sends packets to IOS XR for routing using the routing state in XR Routing Information Base (RIB) or Forwarding Information Base (FIB). The `to_xr` interface does not have an associated IP address. In Linux, most applications expect the outgoing packets to use the IP address of the outgoing interface as the source IP address.

  With the `to_xr` interface, because there is no IP address, a source hint is required. The source hint can be changed to use the IP address another physical interface IP or loopback IP address. In the following example, the source hint is set to 10.1.1.10, which is the IP address of the Hu0_0_0_24 interface. To use the Management port IP address, change the source hint:

  ```
  Router#bash

  [ios:~]$ip route replace default dev to_xr scope link src 192.168.122.22 metric 2048

  [ios:~]$ip route
  default dev to_xr scope link src 192.168.122.22 metric 2048
  10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
  192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
  ```

  With this updated source hint, any default traffic exiting the system uses the Management port IP address as the source IP address.

- **Local or Connected Routes:** The routes are associated with the subnet configured on interfaces. For example, the 10.1.1.0/24 network is associated with the `Hu0_0_0_24` interface, and the 192.168.122.0/24 subnet is associated with the `Mg0_RP0_CPU0` interface .

Configure VRFs in Linux

VRFs configured in IOS XR are automatically synchronized to the kernel. In the kernel, the VRFs appear as network namespaces (netns). For every globally-configured VRF, a Linux network namespace is created. With this capability it is possible to isolate Linux applications or processes into specific VRFs like an out-of-band management VRF and open-up sockets or send or receive traffic only on interfaces in that VRF.

Every VRF, when synchronized with the Linux kernel, is programmed as a network namespace with the same name as a VRF but with the string `vrf` prefixed to it. The default VRF in IOS XR has the name `default`. This name gets programmed as `vrf-default` in the Linux kernel.

The following example shows how to configure a custom VRF `blue`:

Step 1 Identify the current network namespace or VRF.

Example:

```
[ios:~]$ip netns identify $$
vrf-default
global-vrf
```

Step 2 Configure a custom VRF `blue`.

Example:

```
Router#conf t

Router(config)#vrf blue
Router(config-vrf)#commit
```

Step 3 Verify that the VRF `blue` is configured in IOS XR.

Example:

```
Router#show run vrf
vrf blue
!
```

Step 4 Verify that the VRF `blue` is created in the kernel.

Example:

```
Router#bash

[ios:~]$ls -l /var/run/netns
total 0
-r--r--r--. 1 root root 0 Jul 30 04:17 default
-r--r--r--. 1 root root 0 Jul 30 04:17 global-vrf
-r--r--r--. 1 root root 0 Jul 30 04:17 tpnns
-r--r--r--. 1 root root 0 Aug 1 17:01 vrf-blue
-r--r--r--. 1 root root 0 Jul 30 04:17 vrf-default
-r--r--r--. 1 root root 0 Jul 30 04:17 xrnns
```

Step 5 Access VRF `blue` to launch and execute processes from the new network namespace.

Example:

```
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$
[ios:~]$ip netns identify $$
vrf-blue
[ios:~]$
```

Running an **ifconfig** command shows only the default `to-xr` interface because there is no IOS XR interface in this VRF.

```
[ios:~]$ifconfig
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:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 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:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
[ios:~]$
```

Step 6 Configure an interface in the VRF `blue` in IOS XR. This interface will be configured automatically in the network namespace `vrf-blue` in the kernel.

Example:

The following example shows how to configure HundredGigE 0/0/0/24 interface in `vrf-blue` from IOS XR:

```
Router#conf t
Router(config)#int HundredGigE 0/0/0/24
Router(config-if)#no ipv4 address
Router(config-if)#vrf blue
Router(config-if)#ipv4 address 10.1.1.10/24
Router(config-if)#commit
```

Step 7 Verify that the HundredGigE 0/0/0/24 interface is configured in the VRF `blue` in IOS XR.

Example:

```
Router#show run int HundredGigE 0/0/0/24
interface HundredGigE0/0/0/24
vrf blue
ipv4 address 10.1.1.10 255.255.255.0
!
```

Step 8 Verify that the interface is configured in the VRF `blue` in the kernel.

Example:

```
Router#bash
Thu Aug 1 17:09:39.314 UTC
[ios:~]$
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$
[ios:~]$ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 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:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 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:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 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:500
```

```
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
[ios:~]$
```

Open Linux Sockets

The socket entries are programmed into the Local Packet Transport Services (LPTS) infrastructure that distributes the information through the line cards. Any packet received on a line card interface triggers an LPTS lookup to send the packet to the application opening the socket. Because the required interfaces and routes already appear in the kernel, the applications can open the sockets — TCP or UDP.

Step 1 Verify that applications open up sockets.

Example:

```
Router#bash
[ios:~]$nc -l 0.0.0.0 -p 5000 &
[1] 1160
[ios:~]$
[ios:~]$netstat -nlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 0.0.0.0:5000 0.0.0.0:* LISTEN 1160/nc
tcp 0 0 0.0.0.0:57777 0.0.0.0:* LISTEN 14723/emsd
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 8875/ssh_server
tcp6 0 0 :::22 :::* LISTEN 8875/ssh_server
udp 0 0 0.0.0.0:68 0.0.0.0:* 13235/xr_dhcpcd
Active UNIX domain sockets (only servers)
Proto RefCnt Flags Type State I-Node PID/Program name Path
[ios:~]$exit
Logout
Router#
Router#show lpts pifib brief | i 5000
Thu Aug 1 17:16:00.938 UTC
IPv4 default TCP any 0/RP0/CPU0 any,5000 any
Router#
```

Step 2 Verify that the socket is open.

Example:

```
Router#show lpts pifib brief | i 5000
IPv4 default TCP any 0/RP0/CPU0 any,5000 any
```

Netcat starts listening on port 5000, which appears as an IPv4 TCP socket in the netstat output like a typical Linux kernel. This socket gets programmed to LPTS, creating a corresponding entry in the hardware to the lookup tcp port 5000. The incoming traffic is redirected to the kernel of the active RP where the netcat runs.

Send and Receive Traffic

Connect to the nc socket from an external server. For example, the nc socket was started in the `vrf-default` network namespace. So, connect over an interface that is in the same VRF.

```
[root@localhost ~]#nc -vz 192.168.122.22 5000
Ncat: Version 7.50 ( https://nmap.org/ncat )
```

```
Ncat: Connected to 192.168.122.22:5000.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

Manage IOS XR Interfaces through Linux

The Linux system contains a number of individual network namespaces. Each namespace contains a set of interfaces that map to a single interface in the XR control plane. These interfaces represent the exposed XR interfaces (eXI). By default, all interfaces in IOS XR are managed through the IOS XR configuration (CLI or YANG models), and the attributes of the interface (IP address, MTU, and state) are inherited from the corresponding configuration and the state of the interface in XR.

With the new Packet I/O functionality, it is possible to have an IOS XR interface completely managed by Linux. This also means that one or more of the interfaces can be configured to be managed by Linux, and standard automation tools can be used on Linux servers can be used to manage interfaces in IOS XR.

Note Secondary IPv4 addresses cannot be managed by Linux.

Configure an Interface to be Linux-Managed

This section shows how to configure an interface to be Linux-managed.

Step 1 Check the available exposed-interfaces in the system.

Example:

```
Router(config)#linux networking exposed-interfaces interface ?

  Bundle-Ether     Aggregated Ethernet interface(s) | short name is BE
  FiftyGigE        FiftyGigabitEthernet/IEEE 802.3 interface(s) | short name is Fi
  FortyGigE        FortyGigabitEthernet/IEEE 802.3 interface(s) | short name is Fo
  FourHundredGigE  FourHundredGigabitEthernet/IEEE 802.3 interface(s) | short name is FH
  GigabitEthernet  GigabitEthernet/IEEE 802.3 interface(s) | short name is Gi
  HundredGigE      HundredGigabitEthernet/IEEE 802.3 interface(s) | short name is Hu
  Loopback         Loopback interface(s) | short name is Lo
  MgmtEth          Ethernet/IEEE 802.3 interface(s) | short name is Mg
  TenGigE          TenGigabitEthernet/IEEE 802.3 interface(s) | short name is Te
  TwentyFiveGigE   TwentyFiveGigabitEthernet/IEEE 802.3 interface(s) | short name is TF
  TwoHundredGigE   TwoHundredGigabitEthernet/IEEE 802.3 interface(s) | short name is TH
```

Step 2 Configure the interface to be managed by Linux.

Example:

The following example shows how to configure a HundredGigE interface to be managed by Linux:

```
Router#configure
Router(config)#linux networking exposed-interfaces interface HundredGigE 0/0/0/24 linux-managed
Router(config-exi-if)#commit
```

Step 3 View the interface details and the VRF.

Example:

The following example shows the information for HundredGigE interface:

```
Router#show run interface HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
```

```
mtu 4110
vrf blue
ipv4 mtu 4096
ipv4 address 10.1.1.10 255.255.255.0
ipv6 mtu 4096
ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
!
```

Step 4 Verify the configuration in XR.

Example:

The following example shows the configuration for HundredGigE interface:

```
Router#show running-config linux networking

linux networking
 exposed-interfaces
  interface HundredGigE0/0/0/24 linux-managed
  !
 !
!
```

Step 5 Verify the configuration from Linux.

Example:

The following example shows the configuration for HundredGigE interface:

```
Router#bash
Router:Aug 1 17:40:02.873 UTC: bash_cmd[67805]: %INFRA-INFRA_MSG-5-RUN_LOGIN : User vagrant logged
into shell from vty0

[ios:~]$ip netns exec vrf-blue bash

[ios:~]$ifconfig
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:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 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:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

[ios:~]$ifconfig -a
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
BROADCAST MULTICAST MTU:1500 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:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 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:1
```

```
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 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:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Configure New IP address on the Interface in Linux

This section shows how to configure a new IP address on the Linux-managed interface.

Step 1 Configure the IP address on the interface.

Example:

```
[ios:~]$ip addr add 10.1.1.10/24 dev Hu0_0_0_24
[ios:~]$Router:Aug 1 17:41:11.546 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'system'. Use 'show configuration commit changes 1000000021' to view the changes.
```

Step 2 Verify that the new IP address is configured.

Example:

```
[ios:~]$ifconfig Hu0_0_0_24
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
BROADCAST MULTICAST MTU:1500 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:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Configure Custom MTU Setting

This section shows how to bring up the interface and configure a custom MTU in a Linux-managed interface.

Step 1 Configure the MTU setting.

Example:

```
[ios:~]$ifconfig Hu0_0_0_24 up

[ios:~]$Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN : Interface
HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN : Line protocol on
Interface HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:56.448 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000022' to view the changes.
Router:Aug 1 17:41:56.471 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN : Interface
HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:56.484 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN : Line protocol on
Interface HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:58.493 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000023' to view the changes.
```

```
[ios:~]$
[ios:~]$ ip link set dev Hu0_0_0_24 mtu 4096
[ios:~]$
[ios:~]$Router:Aug 1 17:42:46.830 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'system'. Use 'show configuration commit changes 1000000024' to view the changes.
```

Step 2 Verify that the MTU setting has been updated in Linux.

Example:

```
[ios:~]$ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
inet6 addr: fe80::7ae7:e8ff:fed3:20c0/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:4096 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:648 (648.0 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:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 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:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Step 3 Check the effect on the IOS XR configuration with the change in MTU setting on this interface.

Example:

```
Router#show running-config int HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
 mtu 4110
  vrf blue
  ipv4 mtu 4096
  ipv4 address 10.1.1.10 255.255.255.0
  ipv6 mtu 4096
  ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
  !
 !
!
Router#
Router#show ip int br | i HundredGigE0/0/0/24
HundredGigE0/0/0/24 10.1.1.10 Up Up blue
```

The output indicates that the interface acts as a regular Linux interface, and IOS XR configuration receives inputs from Linux.

Configure Traffic Protection for Linux Networking

Traffic protection provides a mechanism to configure Linux firewalls using IOS XR configuration. These rules can be used to restrict traffic to Linux applications. You can restrict traffic to Linux applications using

native Linux firewalls or configuring IOS XR Linux traffic protection. It is not recommended to use both mechanisms at the same time. Any combination of remote address, local address and ingress interface can be specified as rules to either allow or deny traffic. However, at least one parameter must be specified for the traffic protection rule to be valid.

Note If traffic is received on a protocol or port combination that has no traffic protection rules configured, then all traffic is allowed by default.

This example explains how to configure a traffic protection rule on IOS XR to deny all traffic on port 999 except for traffic arriving on interface HundredGigE0/0/0/25.

Step 1 Configure traffic protection rules.

Example:

```
Router(config)#linux networking vrf default address-family ipv4 protection protocol
tcp local-port 999 default-action deny permit hundredgigE0/0/0/25
Router(config)#commit
```

where —

- **address-family:** Configuration for a particular IPv4 or IPv6 address family.
- **protection:** Configure traffic protection for Linux networking.
- **protocol:** Select the supported protocol - TCP or UDP.
- **local-port:** L4 port number to specify traffic protection rules for Linux networking.
- **port number:** Port number ranges from 1 to 65535 or all ports.
- **default-action:** Default action to take for packets matching this traffic protection service.
- **deny:** Drop packets for this service.
- **permit:** Permit packets to reach Linux application for this service.

Step 2 Verify that the traffic protection rule is applied successfully.

Example:

```
Router(config)#show run linux networking
linux networking
 vrf default
  address-family ipv4
   protection
    protocol tcp local-port 999 default-action deny
     permit interface HundredGigE0/0/0/25
     !
    !
   !
 !
```

Communication Outside Cisco IOS XR

Table 3: Feature History Table

Feature Name	Release Information	Description
Virtual IP address in the Linux networking stack	Release 7.5.2	Virtual IP addresses allow a single IP address to connect to the current active RP after an RP switchover event. In addition, this functionality enables your network stack to support virtual IP addresses for third-party applications and IOS XR applications that use the Linux networking stack. The following commands are modified: • ipv4 virtual address • ipv6 virtual address • show kim status

To communicate outside Cisco IOS XR, applications use the `fwdintf` interface address that maps to the `loopback0` interface or a configured Gigabit Ethernet interface address. For information on the various interfaces on IOS XR, see Application Hosting on the Cisco IOS XR Linux Shell, on page 5.

To have an iPerf or Chef client on IOS XR communicate with its respective server outside IOS XR, you must configure an interface address as the source address on XR. The remote servers must configure this route address to reach the respective clients on IOS XR.

Virtual addresses can be configured to access a router from the management network, using the Linux-based app gRPC, through a single virtual IP address. On a device with two or more RPs, the virtual address refers to the management interface that is currently active. This functionality can be used across RP failover without the information of which RP is currently active. This is applicable to the Linux packet path.

This section provides an example of configuring a Gigabit Ethernet interface address as the source address for external communication.

Using a Gigabit Ethernet Interface for External Communication

To configure a GigE interface on IOS XR for external communication, use these steps:

1. Configure a GigE interface.

```
RP/0/RP0/CPU0:ios(config)# interface GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 192.57.43.10 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# no shut
RP/0/RP0/CPU0:ios(config-if)# commit
Fri Oct 30 07:51:14.785 UTC
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# exit
```

2. Verify whether the configured interface is up and operational on IOS XR.

```
RP/0/RP0/CPU0:ios# show ipv4 interface brief
Fri Oct 30 07:51:48.996 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

3. Enter the Linux bash shell and verify if the configured interface is up and running.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash


[xr-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Gi0_0_0_1 Link encap:Ethernet  HWaddr 52:46:2e:49:f6:ff
          inet addr:192.57.43.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:2eff:fe49:f6ff/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:294 (294.0 B)  TX bytes:504 (504.0 B)

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:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:392 (392.0 B)  TX bytes:532 (532.0 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:1482  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:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:672 (672.0 B)  TX bytes:672 (672.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

4. Exit the Linux bash shell and configure the GigE interface as the source address for external communication.

```
[xr-vm_node0_RP0_CPU0:~]$ exit

RP/0/RP0/CPU0:ios# config
Fri Oct 30 08:55:17.992 UTC
RP/0/RP0/CPU0:ios(config)# tpa address-family ipv4 update-source gigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config)# commit
Fri Oct 30 08:55:38.795 UTC
```

Note By default, the `fwdintf` interface maps to the `loopback0` interface for external communication. This is similar to binding a routing process or router ID to the `loopback0` interface. When you use the `tpa address-family ipv4 update-source` command to bind the `fwdintf` interface to a Gigabit Ethernet interface, network connectivity can be affected if the interface goes down.

5. Enter the Linux bash shell and verify whether the GigE interface address is used by the `fwdintf` interface for external communication.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash

[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

External communication is successfully enabled on IOS XR.

East-West Communication for Third-Party Applications

East-West communication on IOS XR is a mechanism by which applications hosted in containers interact with native XR applications (hosted in the XR control plane).

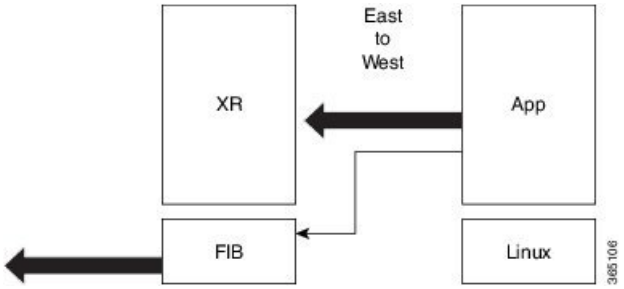
The following figure illustrates how a third-party application hosted on IOS XR interacts with the XR Control Plane.

The application sends data to the Forwarding Information Base (FIB) of IOS XR. The application is hosted in the east portion of IOS XR, while the XR control plane is located in the west region. Therefore, this form of communication between a third-party application and the XR control plane is termed as East-West (E-W) communication.

Third-party applications such as Chef Client and Puppet Agent use this mode of communication to configure and manage containers, packages, and applications on IOS XR. In the future, this support could be extended to IOS XR, configured and managed by such third-party applications.

Note East-West communication is not supported on IOS XR from software release 7.9.1.

Figure 3: East-West Communication on IOS XR



For a third-party application to communicate with IOS XR, the Loopback1 interface must be configured. This is explained in the following procedure.

1. Configure the Loopback1 interface on IOS XR.

```
RP/0/RP0/CPU0:ios(config)# interface Loopback1
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 8.8.8.8/32
RP/0/RP0/CPU0:ios(config-if)# no shut
RP/0/RP0/CPU0:ios(config-if)# commit
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)#
```

2. Verify the creation of the Loopback1 interface.

```
RP/0/RP0/CPU0:ios# show ipv4 interface brief
Thu Nov 12 10:01:00.874 UTC

Interface                  IP-Address      Status              Protocol
Loopback0                  1.1.1.1         Up                  Up
Loopback1                  8.8.8.8         Up                  Up
GigabitEthernet0/0/0/0     192.164.168.10  Up                  Up
GigabitEthernet0/0/0/1     192.57.43.10    Up                  Up
GigabitEthernet0/0/0/2     unassigned      Shutdown            Down
MgmtEth0/RP0/CPU0/0        192.168.122.197 Up                  Up
RP/0/RP0/CPU0:ios#
```

3. Enter the third-party network namespace or global VRF depending on the version of IOS XR version you are using for your network.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash
```

```
/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash
```

4. Verify whether the Loopback1 interface address has been mapped to the E-W interface.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

Configuring Multiple VRFs for Application Hosting

Cisco NCS 540 routers support the configuration of multiple VRFs. The applications hosted in third-party containers can communicate with VRFs configured on XR, after east-west communication has been enabled on the VRFs.

This section describes the configuration for creating mulitple VRFs, and enabling east-west communication between the applications and the VRFs.

Configuration Procedure

Use the following steps to configure multiple VRFs for use on Cisco IOS XR.

1. Configure VRFs on XR.

```
RP/0/RP0/CPU0:ios(config)# vrf purple
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-vrf)# exit

RP/0/RP0/CPU0:ios(config)# vrf green
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-vrf)# exit



RP/0/RP0/CPU0:ios(config)# telnet vrf purple ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf purple ipv6 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf green ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf green ipv6 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet ipv6 server max-servers 2
```

2. Configure the interfaces to be used with the VRFs.

```
RP/0/RP0/CPU0:ios(config)# interface loopback1
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 1.1.1.1 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 10::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface loopback2
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 2.2.2.2 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 20::1/64
RP/0/RP0/CPU0:ios(config-if)# exit
```

```
RP/0/RP0/CPU0:ios(config)# interface loopback3
RP/0/RP0/CPU0:ios(config-if)#vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 3.3.3.3 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 30::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface loopback4
RP/0/RP0/CPU0:ios(config-if)# vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 4.4.4.4 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 40::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface mgmtEth 0/RP0/CPU0/0
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address dhcp
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 10.20.30.40 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 24::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface gigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-if)# vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 40.30.20.10 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 22::1/64
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# commit
Fri Sep  1 12:04:37.796 UTC
```

3. Configure TPA VRFs.

```
RP/0/RP0/CPU0:ios(config)# tpa
RP/0/RP0/CPU0:ios(config-tpa)# vrf purple
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback1
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback2
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source Gigabitethernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# exit


RP/0/RP0/CPU0:ios(config-tpa)# vrf green
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback3
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback4
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source Gigabitethernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# exit
RP/0/RP0/CPU0:ios(config-tpa)# exit
```

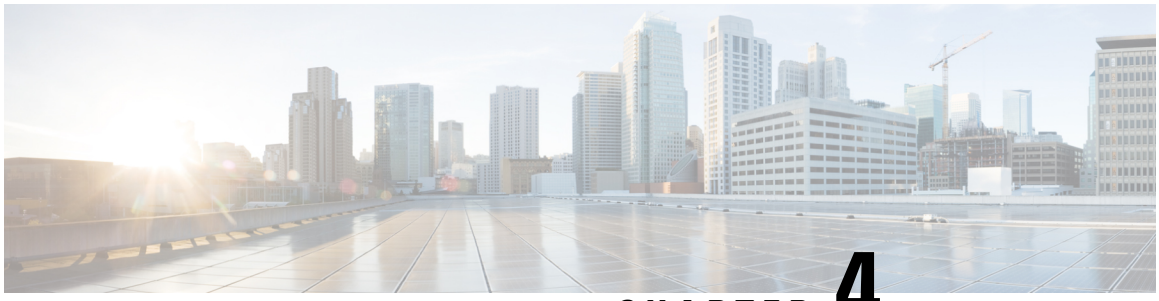
4. Validate the configuration.

```
RP/0/RP0/CPU0:ios(config)# show run
Fri Sep  1 12:06:35.596 UTC
...
```

```
vrf purple
address-family ipv4
address-family ipv6
vrf green
address-family ipv4
address-family ipv6

telnet vrf green ipv4 server max-servers 2
telnet vrf green ipv6 server max-servers 2
telnet vrf purple ipv4 server max-servers 2
telnet vrf purple ipv6 server max-servers 2
telnet vrf default ipv4 server max-servers 2
telnet vrf default ipv6 server max-servers 2
...
!
tpa
 vrf purple
  east-west loopback1
  east-west loopback2
  address-family ipv4
   update-source GigabitEthernet0/0/0/0
 !
  address-family ipv6
   update-source GigabitEthernet0/0/0/0
 !

 vrf green
  east-west loopback3
  east-west loopback4
  address-family ipv4
   update-source GigabitEthernet0/0/0/1
 !
  address-family ipv6
   update-source GigabitEthernet0/0/0/1
 !
!
interface loopback1
 vrf purple
 ipv4 address 1.1.1.1 255.255.255.0
 ipv6 address 10::1/64
!
interface loopback2
 vrf purple
 ipv4 address 2.2.2.2 255.255.255.0
 ipv6 address 20::1/64
!
interface loopback3
 vrf green
 ipv4 address 3.3.3.3 255.255.255.0
 ipv6 address 30::1/64
!
interface loopback4
 vrf green
 ipv4 address 4.4.4.4 255.255.255.0
 ipv6 address 40::1/64
!
interface MgmtEth0/RP0/CPU0/0
 vrf purple
 ipv4 address dhcp
!
router static
 address-family ipv4 unicast
  0.0.0.0/0 MgmtEth0/RP0/CPU0/0 10.0.2.2
```

```
!
!
```

You have successfully configured multiple VRFs for use on Cisco IOS XR.



CHAPTER 4

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.

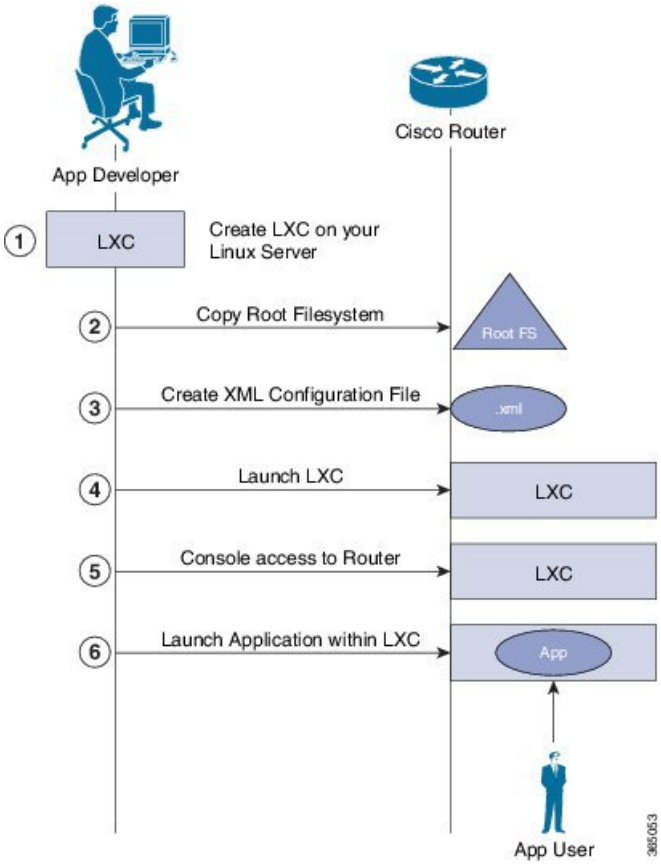
Container Application Hosting

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 59.

Figure 4: 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 5: 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.

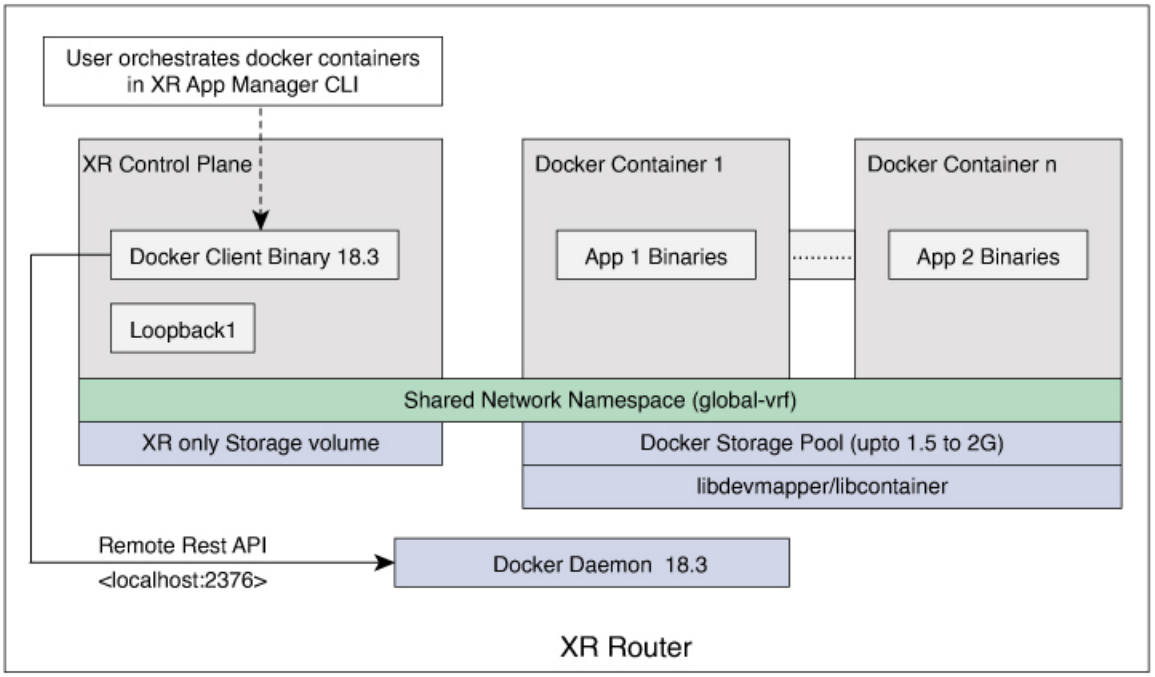
Docker follows a layered approach that consists of a base image at the bottom that supports layers of applications on top. The base images are available publicly in a repository, depending on the type of application you want to install on top. You can manipulate docker images by using the docker index 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 6: Docker Workflow for Updating Applications

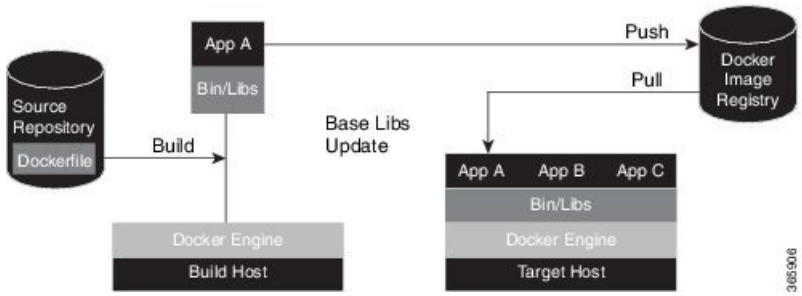


The application binaries for the applications to be hosted are installed inside the docker container.

Hosting Applications in Docker Containers

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

Figure 7: Docker Workflow for Application Hosting

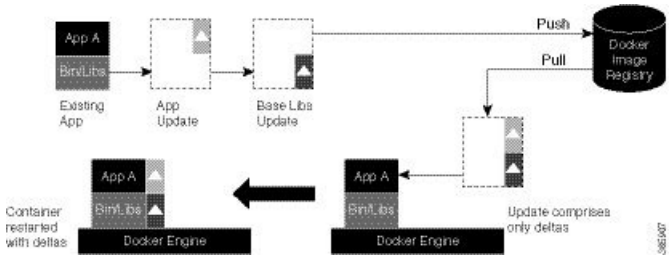


1. The docker file in the source repository is used to build the application binary file on your (docker engine build) host machine.
2. The application binary file is pushed into the docker image registry.
3. The application binary file is pulled from the docker image registry and copied to the docker 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 8: Docker Workflow for Updating Applications



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

Hosting and Seamless Activation of Third Party Applications Using Application Manager

Table 4: 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.
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`

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`

Functionality	Generic Docker Commands	Application Manager Commands
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, on page 53.

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 5: 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 6: 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

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

Table 7: 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

Restrictions 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.

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 8: 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:

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

	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`
		` -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

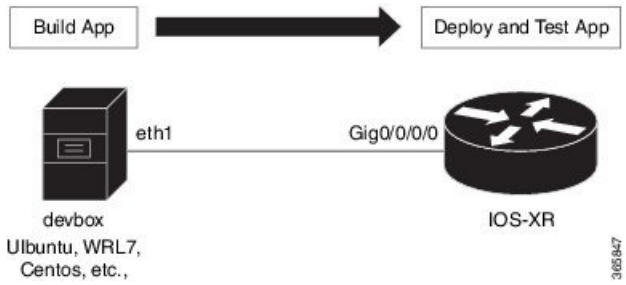
Before you can start using vagrant, ensure that you have fulfilled the following requirements on your host device.

- 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 9: 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 77 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 75, 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.

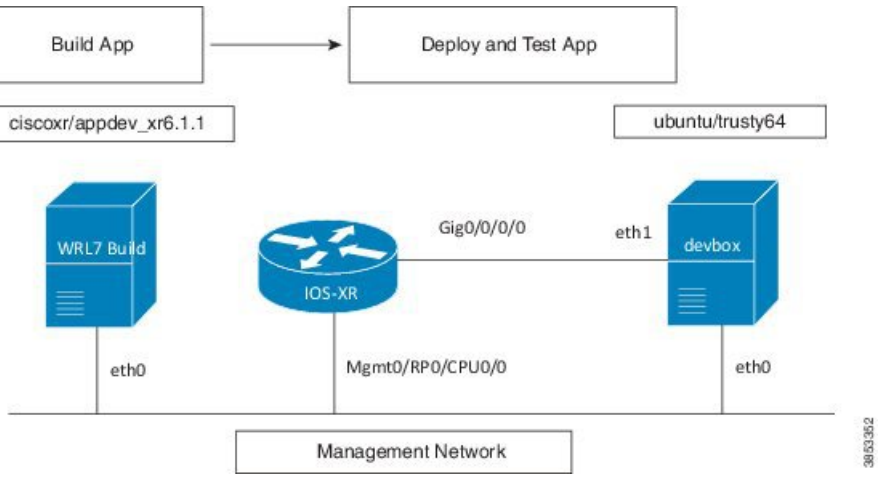
Hosting a Wind River Linux (WRL7) Application Natively By Using Vagrant

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 10: 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 75, 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.

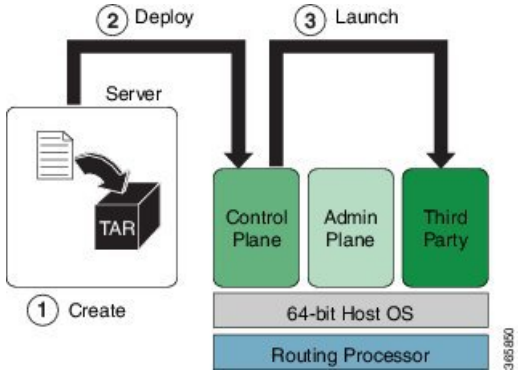
Hosting an Application within a Linux Container (LXC) by Using Vagrant

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 11: 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 75, 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 9: 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 103
- Generate KeyPackage, on page 103
- Onboard Key Package on Router, on page 108
- Generate Signed RPM, on page 110
- Onboard Signed RPM Package on Router, on page 112

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 105
- Key Package Configuration File, on page 107

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 105.

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 10: 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.
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 11: 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.

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.`

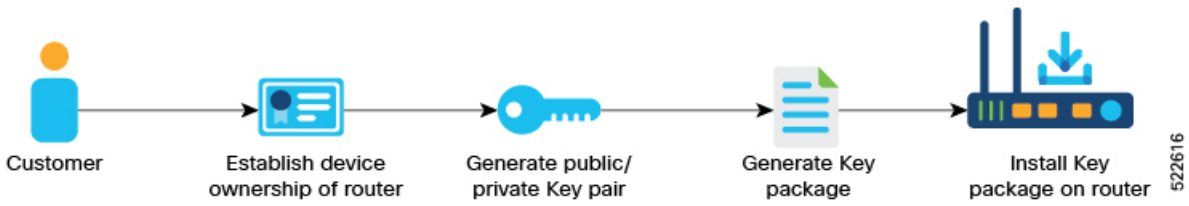
TIMESTAMP	Timestamp in RFC2822 format format	YES	To prevent replay attacks, a key package is one-time use only, which is as per the timestamp available in the key package. RFC2822 format timestamp can be generated by the command **date -R** on Linux devices.

Flag	Possible Values	Mandatory Field	Purpose
KEYTYPE	X509KEY GPGKEY	NO	Defines the type of key being carried in the key package, either X509 or GPG key.
MULTIPLE_KEYPACKAGE	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_KEYPACKAGE 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 12: 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 108.

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 108.

Provisioning Key Packages on the Router

Before you begin

Ensure that your device ownership is established.

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 112.

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.

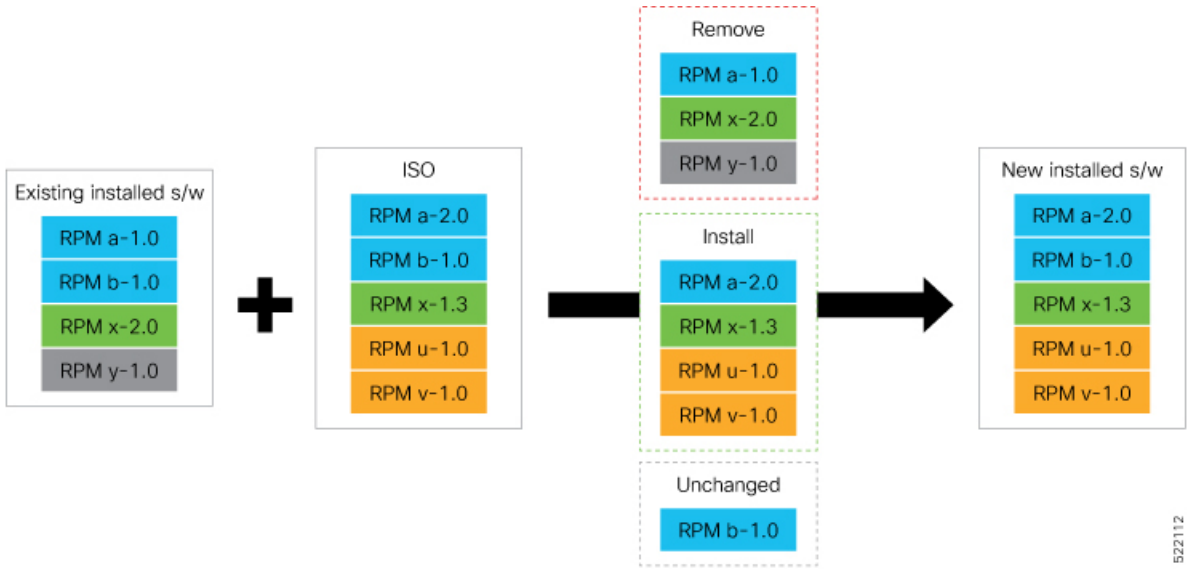
GISO supports automatic dependency management, and provides these functionalities:

- 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.



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 12: 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`

Command	Description

`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

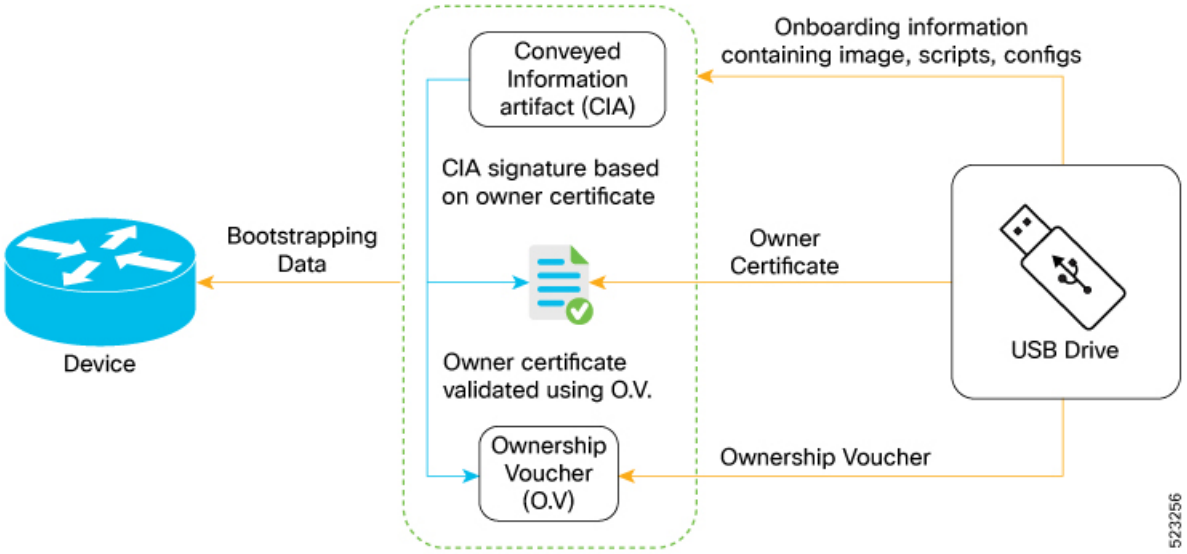
Secure ZTP Work Flow

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 13: 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.

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.

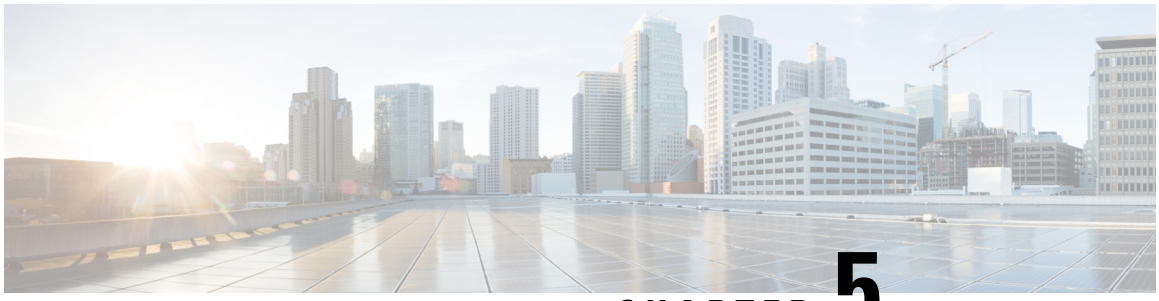
Example:

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

Step 3 Copy the zip file to the router.

Example:

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



CHAPTER 5

Hosting Applications Using Configuration Management Tools

Configuration management tools are used to automate manual tasks, such as setting up servers and network devices. As application delivery requirements keep changing, reconfiguring network equipment becomes a challenge. The manual reconfiguration process is prone to errors, which in turn can cause network outages. Configuration management tools help when configurations need to be updated constantly, and on multiple network devices.

The Cisco IOS XR Software works well with the following configuration management tools:

- Chef
- Puppet

This section explains how you can install, configure, and use the configuration management tools, Chef and Puppet for application hosting on IOS XR.

- Using Chef for Configuring Cisco IOS XR, on page 121
- Using Puppet for Configuring Cisco IOS XR, on page 125
- Using Configuration Management Tools on Vagrant, on page 131

Using Chef for Configuring Cisco IOS XR

Chef is an open-source IT automation tool that you can use to install, configure, and deploy various applications natively on Cisco IOS XR.

To use Chef, you need the following components:

- Chef Client RPM Version 12.5, or later for Cisco IOS XR 6.0
- Chef Server Version 12.4, or higher
- Applications that are compatible with the Wind River Linux 7 environment of IOS XR

You also need three Chef built-in resources to deploy your application natively on IOS XR. The three built-in Chef Resources are:

- Package Resource
- File Resource

- Service Resource

Access the links provided in the following table for additional details on Chef and Chef resources:

Table 13: Chef Resources

Topic	Link
Chef Software, Inc.	https://www.chef.io/
Chef Overview	https://docs.chef.io/chef_overview.html
Package Resource Reference	https://docs.chef.io/resource_package.html
File Resource Reference	https://docs.chef.io/resource_file.html
Service Resource Reference	https://docs.chef.io/resource_service.html
Chef Server Reference	https://docs.chef.io/install_server.html
Chef Client for Native XR Environment	Chef Client

Installing and Configuring the Chef Client

This section describes the procedure for installing the Chef Client on IOS XR.

Prerequisites

Ensure that the following requirements are met before you proceed with installation:

- Your workstation is set up with the Chef repository and the Chef Development Kit.
- Chef Server Version 12.4, or higher is installed and accessible from your Linux box.
- The Chef Server identification files are available.
- You have the right name server and domain name entries configured in the Linux environment (`/etc/resolv.conf`).
- The router is using a valid NTP server.

Configuration Procedure

To install and configure the Chef Client on IOS XR, follow these steps:

1. From your Linux box, access the IOS XR console through SSH, and log in.

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

   You have entered the IOS XR prompt.

2. Enter the third-party network namespace or global VRF, depending on the version of Cisco IOS XR you are using in your network.

   You can verify whether you are in the namespace by viewing the interfaces, as shown here:

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2 or higher, run the following command */
RP/0/RP0/CPU0:ios# bash


[xr-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

3. (Optional) Configure a proxy server (`http_proxy`, `https_proxy`) as needed.

```
http_proxy=http://proxy.youtube.com:8080
https_proxy=https://proxy.youtube.com:8080
```

4. Install the Chef Client.

```
[xr-vm_node0_RP0_CPU0:~]$ yum install https://chef.io/chef/install.sh
```

The Chef **install.sh** script automatically determines the latest version of the Chef Client RPM for installation.

5. Copy the `validation.pem` file from the Chef server to `/etc/chef/validation.pem`
6. Edit the Chef Client configuration file at `/etc/chef/client.rb` with Chef Server identification and Client settings.

```
validation_client_name 'chef-validator'
chef_server_url 'https://my_chef_server.youtube.com/organizations/chef'
node_name 'n3k.youtube.com' # "This" client device.
cookbook_sync_threads 5 # necessary for small memory switches (4G or less)
interval 30 # client-run interval; remove for "never"
```

7. Run the Chef Client.

```
[xr-vm_node0_RP0_CPU0:~]$ chef-client
```

Note To run the Client once, use the **chef-client --once** command. For more information, see the Chef documentation at https://docs.chef.io/chef_client.html

The Chef Client is successfully installed on IOS XR.

Creating a Chef Cookbook with Recipes

A Chef cookbook, loaded with Chef recipes, can be created on your Linux workstation, and copied to the Chef server. After you install the Chef client on IOS XR, the cookbook with recipes can be downloaded from the Chef server, and used while running the client.

Prerequisites

Ensure the following requirements are met before you proceed:

- You have access to the application package compatible with the native IOS XR environment.
- Target application package is hosted on an accessible repository or downloaded to a boot flash.

Configuration Procedure

Use the following procedure to create a Chef recipe that starts the `bootlogd` service, and installs iPerf on IOS XR:

1. Create a cookbook on your Linux workstation by using the corresponding knife command.

```
knife cookbook create cisco-network-chef-cookbook
```

2. Create the Chef recipe file to install iPerf, and add it to the cookbook.

   The Chef recipe must be created in the `cisco-network-chef-cookbook/recipes/` directory. For it to be loaded automatically by the Chef Client, the Chef recipe must be named as `default.rb`.

```
#
# Recipe:: demo_default_providers
#
# Copyright (c) 2015 The Authors, All Rights Reserved.
```

```
package = 'iperf-2.0.5-r0.0.core2_64.rpm'
service = 'bootlogd'

remote_file "/#{package}" do
  source "http://10.105.247.73/wrl7_yum_repo/#{package}"
  action :create
end

yum_package "#{package}" do
  source "/#{package}"
  action :install
end

service "#{service}" do
  action :start
end
```

3. Access the Chef Server from your Linux workstation and upload the cookbook to the server.
4. Log into the IOS XR shell, and run the Chef Client to load and execute the cookbook.

```
[xr-vm_node0_RP0_CPU0:~]$chef-client
```

The iperf RPM is installed on IOS XR.

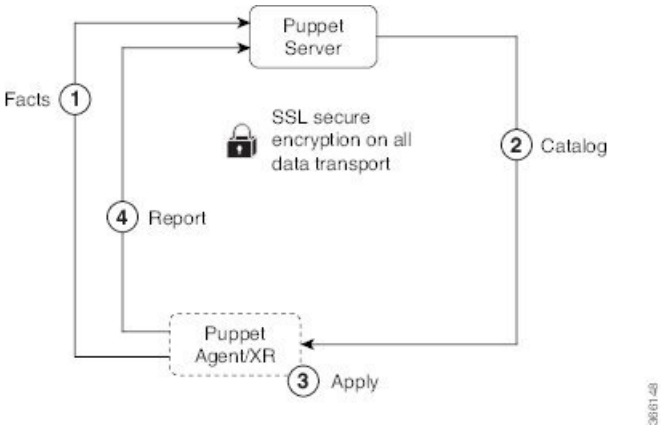
For additional details on the Chef Client, refer to https://docs.chef.io/chef_client.html

Using Puppet for Configuring Cisco IOS XR

Puppet is an open-source configuration management and automation tool that you can use to install and configure various applications on IOS XR. Puppet is provided by Puppet Labs, and runs well on Windows, Unix, and Linux systems. Puppet uses its own declarative langauage to describe system configuration.

Puppet follows a client-server model. The Puppet client is known as the Puppet Agent, and is installed on XR. The configuration file, called the Puppet manifest, is stored on the Puppet Server and contains configuration for multiple nodes. On receiving the information about XR from the Puppet Agent, the Puppet Server compiles the manifest into a catalog that can be used to configure the node that sent the information. This workflow is described in the following illustration.

Figure 14: Basic Puppet Workflow



1. The Puppet Agent sends information about IOS XR to the Puppet Server.
2. The Puppet Server compiles the information into a Catalog, and sends to the Puppet Agent.
3. The Puppet Agent applies the catalog to XR.
4. The Puppet Agent sends a configuration complete report to the Puppet Server.

To use Puppet, you need the following components:

- Puppet RPM built for IOS XR.
- Puppet Server Version 4.0, or higher.

Table 14: Puppet Resources

Topic	Link
Cisco Github Puppet Yang Module	https://github.com/cisco/cisco-yang-puppet-module
Puppet Labs	https://puppetlabs.com/
Package Type Reference	https://docs.puppetlabs.com/references/latest/type.html#package
File Type Reference	https://docs.puppetlabs.com/references/latest/type.html#file
Service Type Reference	https://docs.puppetlabs.com/references/latest/type.html#service
Puppet Server Reference	Puppet Server
Puppet Agent for IOS XR Environment	Puppet Agent

Installing and Configuring the Puppet Agent

This section describes how you can install and configure the Puppet Agent on IOS XR.

Prerequisites

Ensure that the following requirements are met before you proceed with installation.

- Puppet Server Version 4.0, or higher is installed and accessible from your workstation.
- You have the right name server and domain name entries configured in the Linux environment (`/etc/resolv.conf`).
- The router is using a valid NTP server.

Configuration Procedure

To install and configure the Puppet Agent on IOS XR, follow these steps:

1. From your Linux box, access the IOS XR console through SSH, and log in.

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

You have entered the IOS XR prompt.

2. Enter the third-party network namespace or global VRF, depending on the version of Cisco IOS XR you are using in your network.

   You can verify whether you are in the namespace by viewing the interfaces, as shown:

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2 or higher, run the following command */
RP/0/RP0/CPU0:ios# bash

[xr-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

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:1482  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:1500  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)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

3. (Optional) Configure a proxy server (`http_proxy`, `https_proxy`), as needed.

```
http_proxy=http://proxy.youtube.com:8080
https_proxy=https://proxy.youtube.com:8080
```

4. Install the Puppet Agent.

```
[xr-vm_node0_RP0_CPU0:~]$ wget http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
[xr-vm_node0_RP0_CPU0:~]$ wget http://yum.puppetlabs.com/RPM-GPG-KEY-reductive
[xr-vm_node0_RP0_CPU0:~]$ rpm --import RPM-GPG-KEY-puppetlabs RPM-GPG-KEY-reductive
[xr-vm_node0_RP0_CPU0:~]$ yum install
http://yum.puppetlabs.com/puppetlabs-release-pc1-cisco-wrlinux-7.noarch.rpm
```

5. Edit the Puppet Agent configuration file at `/etc/puppetlabs/puppet/puppet.conf` with Puppet Server identification and Agent settings.

The Puppet Agent is successfully installed on IOS XR.

Creating a Puppet Manifest

This section explains how you can create a Puppet manifest on the Puppet Server for installing an application, such as iPerf (RPM file).

Prerequisites

Ensure the following requirements are met before you proceed:

- You have access to the application package compatible with the native IOS XR environment.
- Target application package is hosted on an accessible repository, or downloaded to a boot flash.

Configuration Procedure

To create a sample Puppet manifest to start the `bootlogd` service, and install iPerf on IOS XR, follow these steps:

1. Create a Puppet manifest on Puppet Server to install your application.

   The manifest must be created in the `/etc/puppetlabs/code/environments/production/manifests/` directory. For it to be launched automatically by Puppet Server, the manifest file must be named, `site.pp`.

```
# Manifest to demo builtin providers
#

class ciscopuppet::demo_builtin_providers {

  $package = 'iperf'
  $service = 'bootlogd'

  yumrepo { 'wrl7-repo':
    ensure => present,
    name => 'wrl7-repo',
    baseurl => 'http://10.105.247.73/wrl7_yum_repo/',
    gpgcheck => 0,
    enabled => 1,
    proxy => '_none_',
  }

  package { $package:
    ensure => present,
    require => Yumrepo['wrl7-repo'],
```

```
  }

  service { $service:
    ensure => running,
  }

}

node 'default' {
  include ciscopuppet::demo_builtin_providers
}
```

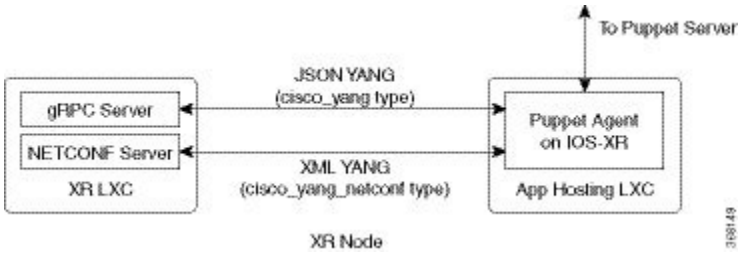
2. Access and trigger the Puppet Agent to converge the system based on the manifest defined on the Puppet Server.

The iPerf RPM is installed on IOS XR by the Puppet Agent.

Using Yang Models with Puppet on IOS XR

You can install the Puppet Agent within a third-party LXC on IOS XR and enable it to interact with the gRPC and Netconf servers installed natively within the XR LXC. The Puppet Agent uses gRPC Ruby libraries to send and receive Yang data in JSON format. The Puppet Agent interacts with the Netconf server to send and receive Yang data in XML format. The workflow is described in the following illustration.

Figure 15: Yang with Puppet Workflow



Installing the Cisco Yang Puppet Module on Puppet Server

Before you can create a sample Puppet Manifest with Yang on the Puppet Server you must install the Cisco Yang Puppet module by executing the following command on the Puppet Server:

```
puppet module install ciscoeng-ciscoyang
```

Alternately, you can manually install the Cisco Yang Puppet module from the github source by using the following commands:

```
git clone https://github.com/cisco/cisco-yang-puppet-module.git
cd cisco-yang-puppet-module
puppet module build
sudo puppet module install pkg/ciscoeng-ciscoyang-1.0.3.tar.gz
```

Sample Puppet Manifest By Using the cisco_yang Type

The following example is a sample manifest that uses the cisco_yang type to configure two VRF instances on IOS XR.

```
node 'default' {
  cisco_yang { 'my-config':
    ensure => present,
    target => '{"Cisco-IOS-XR-infra-rsi-cfg:vrfs": [null]}',
    source => '{"Cisco-IOS-XR-infra-rsi-cfg:vrfs": {
          "vrf":[
            {
                "vrf-name":"VOIP",
                "description":"Voice over IP",
                "vpn-id":{
                  "vpn-oui":875,
                  "vpn-index":3
                },
                "create":[
                  null
                ]
            },
            {
                "vrf-name":"INTERNET",
                "description":"Generic external traffic",
                "vpn-id":{
                  "vpn-oui":875,
                  "vpn-index":22
                },
                "create":[
                  null
                        ]
            }
          ]
      }
    }',
  }
}
```

Sample Puppet Manifest By Using the cisco_yang_netconf Type

The following example is a sample manifest that uses the cisco_yang_netconf type to configure two VRF instances on IOS XR.

```
node 'default' {
  cisco_yang_netconf { 'my-config':
    target => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg"/>',
    source => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
                <vrf>
                  <vrf-name>VOIP</vrf-name>
                  <create/>
                  <description>Voice over IP</description>
                  <vpn-id>
                    <vpn-oui>875</vpn-oui>
                    <vpn-index>3</vpn-index>
                  </vpn-id>
                </vrf>
                <vrf>
                  <vrf-name>INTERNET</vrf-name>
                  <create/>
                  <description>Generic external traffic</description>
                  <vpn-id>
                    <vpn-oui>875</vpn-oui>
                    <vpn-index>22</vpn-index>
                  </vpn-id>
                </vrf>
              </vrfs>',
    mode => replace,
```

```
    force => false,
  }
}
```

For more information on using Yang with Puppet, see https://github.com/cisco/cisco-yang-puppet-module.

Using Configuration Management Tools on Vagrant

You can use vagrant with configuration management tools to automate and execute certain tasks for Cisco IOS XR.

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.

Using Puppet on Vagrant

This section demonstrates how you can use Puppet to configure Cisco IOS XR, by running vagrant on your host device.

Procedure

To start using Puppet on vagrant, use the following 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 `puppet-tutorials/app_hosting/centos-pm` directory for launching the Puppet server.

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

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd puppet-tutorials/app_hosting/centos-pm/

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/puppet-tutorials/app_hosting/centos-pm
 (master)
$ ls
configs/  iosxrv.sh*  puppetmaster.sh*  scripts/
ubuntu-xenial-16.04-cloudimg-console.log  Vagrantfile  xr_config
```

8. Launch the vagrant instance for Puppet server.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/puppet-tutorials/app_hosting/centos-pm
 (master)
$ vagrant up

Bringing machine 'puppetmaster' up with 'virtualbox' provider...
Bringing machine 'iosxrv' up with 'virtualbox' provider...
==> puppetmaster: Box 'bento/centos-7.1' could not be found. Attempting to find and
install...
    puppetmaster: Box Provider: virtualbox
    puppetmaster: Box Version: >= 0
==> puppetmaster: Loading metadata for box 'bento/centos-7.1'
    puppetmaster: URL: https://atlas.hashicorp.com/bento/centos-7.1
==> puppetmaster: Adding box 'bento/centos-7.1' (v2.2.2) for provider: virtualbox
    puppetmaster: Downloading:
https://atlas.hashicorp.com/bento/boxes/centos-7.1/versions/2.2.2/providers/virtualbox.box

    puppetmaster:
==> puppetmaster: Successfully added box 'bento/centos-7.1' (v2.2.2) for 'virtualbox'!
==> puppetmaster: Importing base box 'bento/centos-7.1'...
==> puppetmaster: Matching MAC address for NAT networking...
==> puppetmaster: Checking if box 'bento/centos-7.1' is up to date...
==> puppetmaster: Setting the name of the VM: centos-pm_puppetmaster_1474264139902_14958
==> puppetmaster: Clearing any previously set network interfaces...
==> puppetmaster: Preparing network interfaces based on configuration...
    puppetmaster: Adapter 1: nat
    puppetmaster: Adapter 2: intnet
==> puppetmaster: Forwarding ports...
    puppetmaster: 22 (guest) => 2222 (host) (adapter 1)
==> puppetmaster: Running 'pre-boot' VM customizations...
==> puppetmaster: Booting VM...
==> puppetmaster: Waiting for machine to boot. This may take a few minutes...
    puppetmaster: SSH address: 127.0.0.1:2222
    puppetmaster: SSH username: vagrant
    puppetmaster: SSH auth method: private key
```

```
    puppetmaster: Warning: Remote connection disconnect. Retrying...
...
==> puppetmaster: 127.0.0.1 centos-puppetmaster centos-puppetmaster.cisco.com
==> puppetmaster: 127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
==> puppetmaster: ::1         localhost localhost.localdomain localhost6
localhost6.localdomain6
==> puppetmaster: 10.1.1.20 xr-vm_node0_RP0_CPU0.cisco.com
==> puppetmaster: centos-puppetmaster
==> puppetmaster: Retrieving
https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm
==> puppetmaster: Preparing...
==> puppetmaster: ########################################
==> puppetmaster: Updating / installing...
==> puppetmaster: puppetlabs-release-pc1-1.1.0-2.el7
...

==> iosxrv: Last applied configuration was:
==> iosxrv: Building configuration...
==> iosxrv: !! IOS XR Configuration version = 6.1.1.18I
==> iosxrv: hostname xrv9k
==> iosxrv: domain name cisco.com
==> iosxrv: tpa
==> iosxrv:  address-family ipv4
==> iosxrv:   update-source GigabitEthernet0/0/0/0
==> iosxrv:  !
==> iosxrv: !
==> iosxrv: interface Loopback0
==> iosxrv:  ipv4 address 1.1.1.1 255.255.255.255
==> iosxrv: !
==> iosxrv: interface Loopback1
==> iosxrv:  ipv4 address 10.10.10.10 255.255.255.255
==> iosxrv: !
==> iosxrv: interface GigabitEthernet0/0/0/0
==> iosxrv:  ipv4 address 10.1.1.20 255.255.255.0
==> iosxrv:  no shutdown
==> iosxrv: !
==> iosxrv: router static
==> iosxrv:  address-family ipv4 unicast
==> iosxrv:   0.0.0.0/0 GigabitEthernet0/0/0/0 10.0.0.1
==> iosxrv:  !
==> iosxrv: !
==> iosxrv: ssh server v2
==> iosxrv: ssh server netconf vrf default
==> iosxrv: grpc
==> iosxrv:  port 57777
==> iosxrv: !
==> iosxrv: netconf-yang agent
==> iosxrv:  ssh
==> iosxrv: !
==> iosxrv: end
...
```

9. Create and apply a sample Puppet manifest file.

a. Create and copy sample Puppet manifest file to Puppet Server.

In this example, we use a sample Puppet manifest file to configure two VRF instances on IOS XR by using the Yang Netconf type. . The sample file has already been created and placed in the Puppet-Yang git repository. The file contents are as follows.

```
node 'default' {
  file { "/root/temp/vrfs.json":
    source => "puppet:///modules/ciscoyang/models/defaults/vrfs.json"}
```

```
  # Configure two vrfs (VOIP & INTERNET)
  cisco_yang { '{"Cisco-IOS-XR-infra-rsi-cfg:vrfs": [null]}':
    ensure => present,
    source => '/root/temp/vrfs.json',
  }
}
```

Locate and copy the manifest file.

```
annseque@ANNSEQUE-WS02 MINGW64
~/vagrant-xrdocs/puppet-tutorials/app_hosting/centos-pm (master)
$ vagrant ssh puppetmaster
[vagrant@centos-puppetmaster ~]$ find . -name site.pp
./cisco-yang-puppet-module/examples/site.pp

[vagrant@centos-puppetmaster ~]$ sudo cp ./cisco-yang-puppet-module/examples/site.pp

/etc/puppetlabs/code/environments/production/manifests/
[vagrant@centos-puppetmaster ~]$exit
```

10. Create a `/root/temp` directory on XR (IOS-XRv), which will be used by the Puppet Agent for locating the configuration file.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/puppet-tutorials/app_hosting/centos-pm
 (master)
$ vagrant ssh iosxrv
Last login: Tue Sep 20 03:49:04 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$ sudo mkdir /root/temp/
```

The contents of the configuration file (vrfs.json) for creating two VRF instances is as shown.

```
{
   "Cisco-IOS-XR-infra-rsi-cfg:vrfs":{
      "vrf":[{
            "vrf-name":"VOIP",
            "description":"Voice over IP",
            "vpn-id":{"vpn-oui":87, "vpn-index":3},
            "create":[null]
         },
         {
            "vrf-name":"INTERNET",
            "description":"Generic external traffic",
            "vpn-id":{"vpn-oui":85, "vpn-index":22},
            "create":[null]
         }]
   }
}
```

11. Run the Puppet agent to apply the configuration on XR.

```
xr-vm_node0_RP0_CPU0:~$ sudo puppet agent -t
xr-vm_node0_RP0_CPU0:~$ exit
```

12. Verify if the VRF configuration is successful on XR.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/puppet-tutorials/app_hosting/centos-pm
 (master)
$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:


RP/0/RP0/CPU0:xrv9k# show running-configuration vrf
```

```
vrf VOIP
 description Voice over IP
 vpn id 57:3
!
vrf INTERNET
 description Generic external traffic
 vpn id 55:16
!
$ exit
```

You have successfully used Puppet on vagrant to configure Cisco IOS XR.

Using Ansible for Hosting Applications

Ansible is an automation tool used to configure a device, deploy applications, and manage services. It differs from Chef and Puppet in that it does not need an agent or a client to interact with the Ansible program running on a server.

You can use Ansible to automate tasks on Cisco IOS XR that are time consuming and cumbersome to execute. For instance, you can create an Ansible playbook in YAML with a set of show commands that you need to run at regular intervals. Every time you need to run the set of commands, you can run the playbook with a single command and achieve all the results at once. Alternately, you can create an Ansible module to do a more complex task and invoke it with a playbook.

Ansible Modes of Operation

Ansible module can run on a Linux server or on a router running Cisco IOS XR. When an Ansible module runs on a Linux server, it is considered to be operating in the local mode. When an Ansible module runs on a router, it is considered to be operating in the remote mode.

The two modes operate differently from each other.

To use Ansible in the local or remote mode, use the respective steps described as follows:

- **Ansible in Local Mode**:
  1. Run Ansible program on a Linux server with your Ansible playbook configured to use local mode.
  2. The Ansible playbook invokes the Ansible module to run on the Linux server.
  3. The Ansible module establishes an SSH connection through Port 22 to the router running IOS XR, and executes the specified XR commands.
  4. The command outputs are displayed on the Linux server.

- **Ansible in Remote Mode**:
  1. Configure a router running IOS XR to allow Third Party Network Namespace (TPNNS) shell on Port No. 57722.

     For information on accessing TPNNS, see Accessing the Third-Party Network Namespace on Cisco IOS XR Linux Shell, on page 6.
  2. Configure Ansible to use the SSH Port No. 57722, instead of the default Port No. 22.
  3. Run the Ansible program on a Linux server with your Ansible playbook configured to use remote mode.

4. The Ansible program establishes an SSH connection through Port No. 57722 to TPNNS shell on the router.
5. The Ansible module as defined in the Ansible playbook and other related Ansible system modules are automatically downloaded to the router by the Ansible program.
6. The Ansible module is invoked to run in the TPNNS shell.

   A helper program, such as **/pkg/bin/xr_cli** or **/pkg/sbin/config**, is used by the Ansible module to execute XR commands in the TPNNS shell.
7. The command outputs are sent to the Linux server in JSON format so that it can be displayed on the Linux server.

For information on accessing XR for using Ansible, see the following table.

Table 15: Accessing IOS XR Through Ansible

Mode of Access	Method of Access	SSH Port Number	Helper Programs
Local	Console CLI	22	IOS XR CLI Shell
Local	TPNNS CLI	57722	• /pkg/bin/xr_cli • /pkg/sbin/config
Local	Cisco XML	22	IOS XR XML Agent
Local	Netconf 1.0	22	IOS XR Netconf Agent
Local	Netconf 1.1	830	IOS XR Netconf-yang Agent
Local	YDK Netconf	830	IOS XR Netconf-yang Agent
Remote	TPNNS CLI	57722	• /pkg/bin/xr_cli • /pkg/sbin/config

For more information, see https://github.com/ios-xr/iosxr-ansible.

Using Ansible On Vagrant

This section describes how you can generate a sample Ansible playbook on vagrant.

Sample Ansible Operation on XR

To start using Ansible on XR, 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 75, before proceeding with the following steps.

1. Navigate to the `vagrant-xrdocs/ansible-tutorials/app_hosting/` directory and launch the vagrant instance.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant up
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr' up with 'virtualbox' provider...
==> 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: 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) => 2222 (host) (adapter 1)
==> devbox: Running 'pre-boot' VM customizations...
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2222
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    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/ansible-tutorials/app_hosting
==> 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: Clearing any previously set forwarded ports...
==> rtr: Fixed port collision for 57722 => 2222. Now on port 2200.
==> 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) => 2200 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
    rtr: 58822 (guest) => 58822 (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: 127.0.0.1:2200
    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.

==> 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/ansible-tutorials/app_hosting (master)
$ vagrant status
Current machine states:

devbox                    running (virtualbox)
rtr                       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. Configure access to XR (`rtr`).

   a. Access the `devbox` through SSH and copy its public key by using SCP.

   ```
   annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
   $ vagrant ssh devbox
   Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-92-generic x86_64)

    ...

   7 packages can be updated.
   0 updates are security updates.

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


   Last login: Mon Aug  8 15:16:37 2016 from 10.0.2.2
   vagrant@vagrant-ubuntu-trusty-64:~$ scp -P 57722 /home/vagrant/.ssh/id_rsa.pub
    vagrant@10.1.1.20:/home/vagrant/id_rsa_ubuntu.pub
   id_rsa.pub

   100%  414     0.4KB/s   00:00
   ```

   b. Append the copied keys to `authorized_keys` on XR.

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

   annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
   $ vagrant ssh rtr
   Last login: Mon Aug  8 15:14:31 2016 from 10.1.1.10
   xr-vm_node0_RP0_CPU0:~$ cat /home/vagrant/id_rsa_ubuntu.pub
   >> /home/vagrant/.ssh/authorized_keys
   ```

   By configuring access to XR, Ansible is ready to run without a password.

3. Navigate to `iosxr-ansible/remote/samples` directory to see sample Ansible playbooks that you can run on devbox.

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

---------------------------------------------------------------------------------------------
```

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-92-generic x86_64)

...
35 packages can be updated.
24 updates are security updates.

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


Last login: Tue Sep  6 09:54:13 2016 from 10.0.2.2

------------------------------------------------------------------
vagrant@vagrant-ubuntu-trusty-64:~$ pwd
/home/vagrant

vagrant@vagrant-ubuntu-trusty-64:~$ cd iosxr-ansible/remote/samples
vagrant@vagrant-ubuntu-trusty-64:~/iosxr-ansible/remote/samples$ ls
ifconfig.yml         iosxr_cli.yml         iosxr_install_package.yml  iosxr_rollback.yml
        iosxr_user_remove.yml   show_config_commit   show_users
install              iosxr_get_config.yml  iosxr_reload.yml
iosxr_update_package.yml  iosxr_user_replace.yml  show_install_active
iosxr_clear_log.yml  iosxr_get_facts.yml   iosxr_remove_package.yml   iosxr_user_add.yml
        README.md               show_int_brief
```

4. Run a sample Ansible playbook to view the required information about XR (`rtr`).

```
vagrant@vagrant-ubuntu-trusty-64:~/iosxr-ansible/remote$
ansible-playbook samples/iosxr_cli.yml -e 'cmd="show interface brief"' --become

PLAY [ss-xr] *******************************************************************

TASK [iosxr_cli] ***************************************************************
ok: [10.1.1.20]

TASK [debug] *******************************************************************
ok: [10.1.1.20] => {
    "output.stdout_lines": [
        "",
       "------------------------------ show interface brief ------------------------------",

        "",
        "               Intf       Intf        LineP              Encap  MTU        BW",

        "              Name       State       State              Type (byte)    (Kbps)",


"--------------------------------------------------------------------------------",
        "                Nu0          up          up               Null  1500
0",
        "         Gi0/0/0/0          up          up              ARPA  1514    1000000",

        "     Mg0/RP0/CPU0/0          up          up              ARPA  1514    1000000",

        ""
    ]
}

PLAY RECAP *********************************************************************
```

```
10.1.1.20                  : ok=2    changed=0    unreachable=0    failed=0
```

You have successfully used Ansible on vagrant.

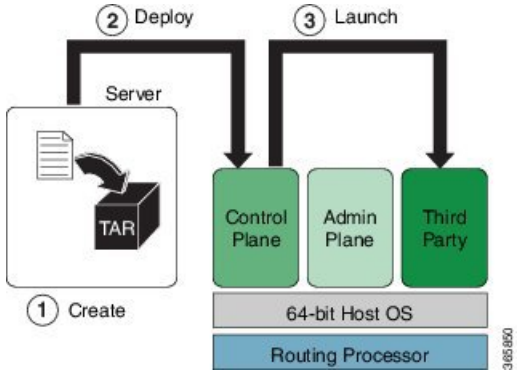
Launching a Linux Container (LXC) By Using Ansible on Vagrant

This section describes how you can launch your own container (LXC) by using Ansible on 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 16: 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 launch your LXC container by using Ansible on a vagrant box, 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 75, before proceeding with the following steps.

1. Navigate to the `vagrant-xrdocs/ansible-tutorials/app_hosting/` directory and launch the vagrant instance.

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

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant up
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr' up with 'virtualbox' provider...
==> 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: 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) => 2222 (host) (adapter 1)
==> devbox: Running 'pre-boot' VM customizations...
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2222
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    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/ansible-tutorials/app_hosting
==> 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: Clearing any previously set forwarded ports...
==> rtr: Fixed port collision for 57722 => 2222. Now on port 2200.
==> 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) => 2200 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
    rtr: 58822 (guest) => 58822 (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: 127.0.0.1:2200
    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.

==> 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/ansible-tutorials/app_hosting (master)
$ vagrant status
Current machine states:
```

```
devbox                    running (virtualbox)
rtr                       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. Configure access to XR (`rtr`), if not done already.

   a. Access the `devbox` through SSH and copy its public key by using SCP.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting
(master)
$ vagrant ssh devbox
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-92-generic x86_64)

 ...

7 packages can be updated.
0 updates are security updates.

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


Last login: Mon Aug  8 15:16:37 2016 from 10.0.2.2
vagrant@vagrant-ubuntu-trusty-64:~$ scp -P 57722 /home/vagrant/.ssh/id_rsa.pub
 vagrant@10.1.1.20:/home/vagrant/id_rsa_ubuntu.pub
id_rsa.pub

100%  414     0.4KB/s   00:00
```

   b. Append the copied keys to `authorized_keys` on XR.

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

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting
(master)
$ vagrant ssh rtr
Last login: Mon Aug  8 15:14:31 2016 from 10.1.1.10
xr-vm_node0_RP0_CPU0:~$ cat /home/vagrant/id_rsa_ubuntu.pub
>> /home/vagrant/.ssh/authorized_keys
```

   By configuring access to XR, Ansible is ready to run without a password.

3. 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**

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

 ...

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) ...

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

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. 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
```

8. 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
```

9. Assume the root user role.

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

10. 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 -czf
xr-lxc-app-rootfs.tar.gz *
```

```
tar: dev/log: socket ignored
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/xr-lxc-app/rootfs#
```

11. 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#
```

12. 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.

13. 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:~#
```

14. Run the Ansible playbook that automatically runs the following steps in deploying an LXC container on XR (`rtr`).

    a. Copies the `xr-lxc-app-xml` file to XR.

    b. Copies the `xr-lxc-app-rootfs.tar.gz` tar ball to XR.

    c. Creates the `xr-lxc-app/rootfs` directory on XR.

    d. Untars the `rootfs` tar ball in the `xr-lxc-app/rootfs` directory.

    e. Verifies if your LXC container is installed on XR. (If not, creates the container by using the **virsh** command.)

    f. Uses the **virsh** command to verify that your LXC container is up and running.

```
root@vagrant-ubuntu-trusty-64:~# exit
exit
------------------------------------------------------------------------------------------------

vagrant@vagrant-ubuntu-trusty-64:~$ ansible-playbook deploy_container.yml

PLAY [ss-xr] ***********************************************************************

TASK [setup] ***********************************************************************
ok: [10.1.1.20]

TASK [Copy XML file] ***************************************************************
ok: [10.1.1.20]

TASK [Copy rootfs tar ball] ********************************************************
ok: [10.1.1.20]

TASK [Create rootfs directory] *****************************************************
ok: [10.1.1.20]

TASK [debug] ***********************************************************************
ok: [10.1.1.20] => {
    "output.stdout_lines": []
}

TASK [grep] ************************************************************************
fatal: [10.1.1.20]: FAILED! => {"changed": true, "cmd": "sudo -i virsh list | grep
xr-lxc-app", "delta": "0:00:01.497387", "end": "2016-09-06 05:49:46.886749", "failed":
 true, "rc": 1, "start": "2016-09-06 05:49:45.389362", "stderr": "", "stdout": "",
"stdout_lines": [], "warnings": []}
...ignoring

TASK [debug] ***********************************************************************
ok: [10.1.1.20] => {
    "output.stdout_lines": []
}
```

```
TASK [virsh create] ************************************************************
changed: [10.1.1.20]

TASK [debug] *******************************************************************
ok: [10.1.1.20] => {
    "output.stdout_lines": [
        "Domain xr-lxc-app created from /home/vagrant/xr-lxc-app.xml"
    ]
}

TASK [command] *****************************************************************
changed: [10.1.1.20]

TASK [debug] *******************************************************************
ok: [10.1.1.20] => {
    "output.stdout_lines": [
        " Id    Name                           State",
        "----------------------------------------------------",
        " 4903  sysadmin                       running",
        " 12021 default-sdr--1                 running",
        " 18703 xr-lxc-app                     running"
    ]
}

PLAY RECAP *********************************************************************
10.1.1.20                  : ok=12   changed=2    unreachable=0    failed=0
```

If for some reason, Ansible playbook does not run, then reapply the environment variables listed in the `ansible_env` file, as shown, and try again.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat iosxr-ansible/remote/ansible_env
export BASEDIR=/home/vagrant
export IOSXRDIR=$BASEDIR/iosxr-ansible
export ANSIBLE_HOME=$BASEDIR/ansible
export ANSIBLE_INVENTORY=$IOSXRDIR/remote/ansible_hosts
export ANSIBLE_LIBRARY=$IOSXRDIR/remote/library
export ANSIBLE_CONFIG=$IOSXRDIR/remote/ansible_cfg
export YDK_DIR=$BASEDIR/ydk/ydk-py
export PYTHONPATH=$YDK_DIR


vagrant@vagrant-ubuntu-trusty-64:~$ export BASEDIR=/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~$ export IOSXRDIR=$BASEDIR/iosxr-ansible
vagrant@vagrant-ubuntu-trusty-64:~$ export ANSIBLE_HOME=$BASEDIR/ansible
vagrant@vagrant-ubuntu-trusty-64:~$ export
ANSIBLE_INVENTORY=$IOSXRDIR/remote/ansible_hosts
vagrant@vagrant-ubuntu-trusty-64:~$ export ANSIBLE_LIBRARY=$IOSXRDIR/remote/library
vagrant@vagrant-ubuntu-trusty-64:~$ export ANSIBLE_CONFIG=$IOSXRDIR/remote/ansible_cfg
vagrant@vagrant-ubuntu-trusty-64:~$ export YDK_DIR=$BASEDIR/ydk/ydk-py
vagrant@vagrant-ubuntu-trusty-64:~$ export PYTHONPATH=$YDK_DIR

-----------------------------------------------------------------------------------------
vagrant@vagrant-ubuntu-trusty-64:~$ ansible-playbook deploy_container.yml

PLAY [ss-xr] *******************************************************************

TASK [setup] *******************************************************************
ok: [10.1.1.20]
...
```

You have successfully launched your LXC by using Ansible on vagrant.

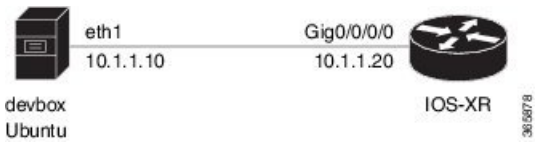
Using Netmiko and Napalm on Vagrant

You can use configuration management tools such as Netmiko and Napalm to manage and monitor a router running Cisco IOS XR. This section describes how you can get started with using Netmiko and Napalm on vagrant.

Topology

The topology used in this example is illustrated in the following figure.

Figure 17: Topology for Netpalm and Netmiko



Procedure for Using Netmiko

To start using Netmiko for managing XR, use the following 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/ansible-tutorials` directory and launch the vagrant instance.

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

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs (master)
$ cd ansible-tutorials/app_hosting/


annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials (master)
$ vagrant up
...
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: 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/ansible-tutorials (master)
$ vagrant status
Current machine states:


devbox                    running (virtualbox)
rtr                       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`.
```

8. Access `devbox` using SSH, and install the netmiko module as root (sudo) user.

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

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

  System information as of Mon Sep 26 05:59:20 UTC 2016

 ...
vagrant@vagrant-ubuntu-trusty-64:~$ sudo pip install netmiko
...
Requirement already satisfied (use --upgrade to upgrade): netmiko in
/usr/local/lib/python2.7/dist-packages
Requirement already satisfied (use --upgrade to upgrade):
paramiko>=1.13.0 in /usr/local/lib/python2.7/dist-packages/paramiko-2.0.2-py2.7.egg
(from netmiko)
Requirement already satisfied (use --upgrade to upgrade):
scp>=0.10.0 in /usr/local/lib/python2.7/dist-packages (from netmiko)
...
```

9. Run python interpreter to verify successful installation of the netmiko module.

Press **CTRL+Z** to exit the interpreter.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import netmiko
```

10. Create your netmiko configuration file by using vi editor.

A sample netmiko configuration file is as shown.

The sample netmiko configuration file displays the interfaces on XR, changes the hostname to 'my_sweet_rtr', commits the host name and displays the host name.

```
vagrant@vagrant-ubuntu-trusty-64:~$ vi netmiko_tut.py
from netmiko import ConnectHandler

cisco_ios_xrv = {
    'device_type': 'cisco_xr',
    'ip':   '10.1.1.20',
    'username': 'vagrant',
    'password': 'vagrant',
    'port' : 22,          # optional, defaults to 22
    'secret': 'secret',     # optional, defaults to ''
    'verbose': False,       # optional, defaults to False
}

net_connect = ConnectHandler(**cisco_ios_xrv)

output = net_connect.send_command('show ip int brief')
print(output)

output = net_connect.send_config_set(['hostname my_sweet_rtr', 'commit'])
print(output)

output = net_connect.send_config_set(['show run | b hostname'])
print(output)
```

Enter **:wq** to save the file and exit the vi editor

11. Use python to execute the netmiko configuration file on XR.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python netmiko_tut.py

Fri Jul 15 12:29:07.691 UTC

Interface                      IP-Address      Status          Protocol Vrf-Name
GigabitEthernet0/0/0/0         10.1.1.20       Up              Up       default
MgmtEth0/RP0/CPU0/0            10.0.2.15       Up              Up       default

config term
Fri Jul 15 12:29:09.739 UTC
RP/0/RP0/CPU0:my_sweetest_rtr(config)#hostname my_sweetest_rtr
RP/0/RP0/CPU0:my_sweetest_rtr(config)#commit
Fri Jul 15 12:29:10.332 UTC
end
config term
Fri Jul 15 12:29:12.475 UTC
RP/0/RP0/CPU0:my_sweetest_rtr(config)#show run | include hostname
Fri Jul 15 12:29:13.052 UTC
Building configuration...
```

```
hostname my_sweetest_rtr
RP/0/RP0/CPU0:my_sweetest_rtr(config)#
```

12. (Optional) To use a more serious application of netmiko, you can use the following steps.

    a. Create a telemetry configuration file by using the vi editor.

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

telemetry
 encoder json
  policy group FirstGroup
   policy test
    transport tcp
    !
   destination ipv4 10.1.1.10 port 2103
commit
```

    Enter **:wq** to save the file and exit the vi editor.

    b. Update the netmiko configuration file to display the contents of the telemetry configuration file.

```
vagrant@vagrant-ubuntu-trusty-64:~$ vi netmiko_tut.py
from netmiko import ConnectHandler

cisco_ios_xrv = {
    'device_type': 'cisco_xr',
    'ip':   '10.1.1.20',
    'username': 'vagrant',
    'password': 'vagrant',
    'port' : 22,          # optional, defaults to 22
    'secret': 'secret',     # optional, defaults to ''
    'verbose': False,       # optional, defaults to False
}

net_connect = ConnectHandler(**cisco_ios_xrv)

output = net_connect.send_command('show ip int brief')
print(output)

output = net_connect.send_config_set(['hostname my_sweet_rtr', 'commit'])
print(output)

output = net_connect.send_config_set(['show run | b hostname'])
print(output)

with open('tel_conf') as f:
    lines = f.read().splitlines()
print lines

tel_out =  net_connect.send_config_set(lines)
print tel_out
```

    c. Use python to execute the updated netmiko configuration file.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python netmiko_tut.py
config term
Thu Jul 14 23:49:25.447 UTC
RP/0/RP0/CPU0:xr(config)#telemetry
RP/0/RP0/CPU0:xr(config-telemetry)# encoder json
RP/0/RP0/CPU0:xr(config-telemetry-json)#  policy group FirstGroup
RP/0/RP0/CPU0:xr(config-policy-group)#   policy test
RP/0/RP0/CPU0:xr(config-policy-group)#    transport tcp
RP/0/RP0/CPU0:xr(config-telemetry-json)#    !
```

```
RP/0/RP0/CPU0:xr(config-telemetry-json)#   destination ipv4 10.1.1.10 port 2103
RP/0/RP0/CPU0:xr(config-policy-group)# commit
...
```

Exit and navigate to the `/vagrant-xrdocs/ansible-tutorials` directory.

d. Access `rtr` through SSH and verify if the telemetry configuration is present.

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials (master)
$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:



RP/0/RP0/CPU0:my_sweet_rtr# show run | begin telemetry
Thu Jul 14 20:58:19.116 UTC
Building configuration...
xml agent ssl
!
xml agent tty
!
telemetry
 encoder json
  policy group FirstGroup
   policy test
   transport tcp
   !
   destination ipv4 10.1.1.10 port 2103
  !
 !
!
end
```

You have successfully used netmiko on vagrant for managing Cisco IOS XR.

Procedure for Napalm

To start using napalm for monitoring XR, use the following steps.

1. Follow Steps 1-7 described in the *Procedure for Netmiko* section.
2. Access `devbox` using SSH, and install the napalm module as root (sudo) user.

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

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

  System information as of Mon Sep 26 05:59:20 UTC 2016

 ...

vagrant@vagrant-ubuntu-trusty-64:~$ sudo pip install napalm
...
Requirement already satisfied (use --upgrade to upgrade): napalm in
/usr/local/lib/python2.7/dist-packages
Requirement already satisfied (use --upgrade to upgrade):
napalm-base in /usr/local/lib/python2.7/dist-packages (from napalm)
Requirement already satisfied (use --upgrade to upgrade):
napalm-eos in /usr/local/lib/python2.7/dist-packages (from napalm)
...
```

3. Run python interpreter to verify successful installation of the napalm module.

   Press **CTRL+Z** to exit the interpreter.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import napalm
```

4. Create your napalm configuration file by using vi editor.

   A sample napalm configuration file is as shown.

   The sample napalm configuration file displays the GigE interfaces on XR with the counters and user information.

```
vagrant@vagrant-ubuntu-trusty-64:~$ vi napalus.py

from napalm import get_network_driver

driver = get_network_driver('iosxr')

device = driver('10.1.1.20', 'vagrant', 'vagrant')

device.open()
# print device.get_facts() ## doesn't work

print device.get_interfaces()
print ''
print device.get_interfaces_counters()
print ''
print device.get_users()

device.close()
```

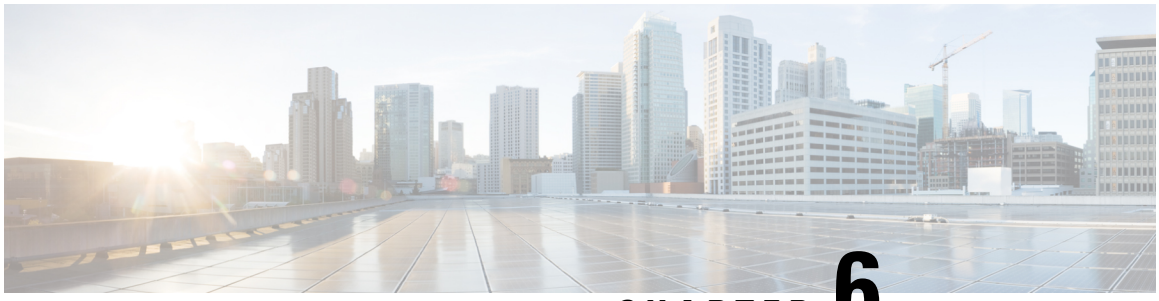
   Enter **:wq** to save the file and exit the vi editor

5. Use python to execute the updated napalm configuration file.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python napalus.py

{
    'GigabitEthernet0/0/0/0': {
        'is_enabled': True,
        'description': u '',
        'last_flapped': -1.0,
        'is_up': True,
        'mac_address': u '0800.27b2.5406',
        'speed': 1000
    }
}

{
    'GigabitEthernet0/0/0/0': {
        'tx_multicast_packets': 0,
        'tx_discards': 0,
        'tx_octets': 6929839,
        'tx_errors': 0,
        'rx_octets': 586788,
        'tx_unicast_packets': 10799,
        'rx_errors': 0,
        'tx_broadcast_packets': 0,
        'rx_multicast_packets': 0,
        'rx_broadcast_packets': 3,
```

```
        'rx_discards': 0,
        'rx_unicast_packets': 9421
    }
}

{
    u 'vagrant': {
        'password': '',
        'sshkeys': [],
        'level': 15
    }
}
```

You have successfully used napalm on vagrant for monitoring Cisco IOS XR.



CHAPTER 6

Cisco Secure DDoS Edge Protection

Table 16: Feature History Table

Feature Name	Release Information	Description
Cisco Secure DDoS Edge Protection	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]) We have now moved DDoS protection to the network edge, ensuring you can mitigate any DDoS attacks at the ingress points and minimize the impact of such attacks on your network and applications running on it. This solution deploys a centralized controller application that manages a distributed network of edge detectors that analyze and mitigate threats across networks.

The Cisco Secure DDoS Edge Protection software solution stops DDoS attacks at the ingress side of the network.

The DDoS Edge Protection solution helps you detect DDoS attacks and take mitigation actions on the router. To enable detection services at the core network, you need to configure the following entities:

- DDoS Edge Protection Controller: This entity manages and monitors the Detector docker application, mitigates attacks, and oversees a distributed network of edge detectors. It analyzes detection trends across the network, orchestrates cross-network visibility and mitigation, and provides complete system management for the entire service.
- DDoS Edge Protection Detector: This entity is a real-time DDoS detection microservice container application that runs as a docker-application on a router with the DDoS controller. The DDOS controller can run on a cloud, server, or customer premises and is connected to this application.

The DDoS Edge Protection supports DDoS detection of both IPv4 and IPv6 traffic. You can choose the interface on which the traffic should be monitored. When the protection software solution is implemented, it filters the IPv4/IPv6 traffic flow and detects DDoS attacks.

Once a DDoS attack is detected, the DDoS Edge Protection Controller initiates a mitigation action, specifying the necessary steps to counteract the attack. This includes enabling traffic classification (TC) as part of the mitigation measures, implementation of rate limiting and so on.

Supported Routers

Cisco Secure DDoS Edge Protection is supported on the following hardware:

- NCS-55A1-48Q6H
- NCS-55A1-48Q6H-SE
- NCS-55A1-48Q-DTC
- NCS-57D2-18DD-S
- NCS-57C3-MOD-S
- NCS-57C3-MOD-SE-S
- NCS-55A1-36H-SE-S
- NCS-55A1-36H-DTC
- NCS-55A1-36H-GLE
- NCS-55A1-36H-S
- NCS-55A2-MOD-SE-S
- NCS-55A2-MOD-HD-S
- NCS-55A2-MOD-SYS
- NCS-55A2-MOD-HX-S
- NCS-55A2-MOD-SE-H-S
- NCS-55A1-24H
- NCS-57B1-6D24H-S
- NCS-57B1-5D24H-SE
- NCS-5501
- NCS-5501-SE
- NCS-55A1-24Q6H-S
- NCS-55A1-24Q-DTCR
- NCS-55A1-24Q-RPHY
- NCS-55A1-24Q6H-SS
- NCS-57C1-48Q6D-S
- NCS-5502-SE
- NCS-5502-U100

Guidelines for Installing DDoS Edge Protection

- Configure the management interface to reach the DDoS controller IP address.
- Manually configure the base ACL, UDF, NetFlow, and SSH configurations.

  For more information, see .
- Reload the router as a hw-module profile configuration is being performed.

Restrictions of DDoS Edge Protection Solution

- Only IPv4 and IPv6 traffic is supported.
- Only default VRF configuration is supported and is limited to the management port. To ensure smooth communication between the Docker and the controller, make sure to set up the management port exclusively in the default VRF.

Install and Configure DDoS Edge Protection

You can install the DDoS Edge Protection application through the DDoS edge protection controller. Perform the following:

1. Install and download the DDoS Edge Protection Controller Software package from the Software Download page. You can access the user interface, when the controller installation is complete.

   Log in to the controller services instance to monitor, manage, and control the device.
2. Perform the following base configurations such as ACL, UDF, hw-module, NetFlow configuration, and SSH manually on the router:

   Configure UDF

```
RP/0/RP0/CPU0:ios(config)#udf udf-ident header outer l3 offset 4 length 2
RP/0/RP0/CPU0:ios(config)#udf udf-chksum header outer l4 offset 16 length 2
RP/0/RP0/CPU0:ios(config)#udf udf-seqnum header outer l4 offset 4 length 4
```

   The user-defined field, allows you to define a custom key by specifying the location and size of the field to match.

   Configure the hardware module or TCAM

```
RP/0/RP0/CPU0:ios(config)#hw-module profile tcam format access-list ipv4 src-addr dst-addr
 src-port dst-port proto tcp-flags packet-length frag-bit precedence enable-capture
ttl-match udf1 udf-chksum udf2 udf-seqnum udf3 udf-ident
```

```
RP/0/RP0/CPU0:ios(config)#hw-module profile tcam format access-list ipv6 src-port dst-addr
 dst-port next-hdr tcp-flags payload-length ttl-match
```

Reload the router (as hw-module profile and UDF configuration is performed).

Configure Loopback

```
RP/0/RP0/CPU0:ios(config)#interface Loopback100
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 15.1.1.2 255.255.255.255
RP/0/RP0/CPU0:ios(config)#interface Loopback101
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 17.1.1.2 255.255.255.255
RP/0/RP0/CPU0:ios(config-if)#
```

Configure Netflow

```
//Configuring Monitor Map
RP/0/RP0/CPU0:ios(config)#flow monitor-map DetectPro_Monitor_IPV6
RP/0/RP0/CPU0:ios(config)# record ipv6 extended
RP/0/RP0/CPU0:ios(config)#exporter DetectPro_GPB
RP/0/RP0/CPU0:ios(config)# cache entries 1000000
RP/0/RP0/CPU0:ios(config)#cache entries active 1
RP/0/RP0/CPU0:ios(config)#cache entries inactive 1
RP/0/RP0/CPU0:ios(config)#cache timeout inactive 1
RP/0/RP0/CPU0:ios(config)#cache timeout rate-limit 1000000
!
RP/0/RP0/CPU0:ios(config)#flow monitor-map DetectPro_Monitor_IPV4
RP/0/RP0/CPU0:ios(config)# record ipv4 extended
RP/0/RP0/CPU0:ios(config)#exporter DetectPro_GPB
RP/0/RP0/CPU0:ios(config)# cache entries 1000000
RP/0/RP0/CPU0:ios(config)#cache entries active 1
RP/0/RP0/CPU0:ios(config)#cache entries inactive 1
RP/0/RP0/CPU0:ios(config)#cache timeout inactive 1
RP/0/RP0/CPU0:ios(config)#cache timeout rate-limit 1000000
!
//Configuring Exporter Map
RP/0/RP0/CPU0:ios(config)#flow exporter-map DetectPro_GPB
RP/0/RP0/CPU0:ios(config)#version protobuf
RP/0/RP0/CPU0:ios(config)#transport udp 5005
RP/0/RP0/CPU0:ios(config)#source TenGigE0/0/0/16
RP/0/RP0/CPU0:ios(config)#destination 15.1.1.2
!
//Configuring Sampler Map
RP/0/RP0/CPU0:ios(config)#sampler-map DetectPro_NFv9
RP/0/RP0/CPU0:ios(config)#random 1 out-of 100
!
```

Configure ACL

```
RP/0/RP0/CPU0:ios(config)#ipv4 access-list myACL
RP/0/RP0/CPU0:ios(config-ipv4-acl)# 1301 permit ipv4 any any
!
RP/0/RP0/CPU0:ios(config)#ipv4 access-list myACL
RP/0/RP0/CPU0:ios(config-ipv6-acl)# 1301 permit ipv6 any any
!
```

For more information on implementing access lists and prefix lists, see Understanding Access-List.

If there is any DDoS attack, the controller performs the mitigation action using the ACL rule automatically.

The following is a sample configuration to deny DDoS attacker traffic using user defined ACE rule:

```
1 deny udp any eq 19 host 45.0.0.1 eq 0 packet-length eq 128 ttl eq 64
2 deny tcp any host 45.0.0.1 eq www match-all -established -fin -psh +syn -urg
packet-length eq 60 ttl eq 64
1301 permit ipv4 any any
```

Configuration updates are sent by the controller to the router.

Configure SSH

```
RP/0/RP0/CPU0:ios(config)#ssh server v2
RP/0/RP0/CPU0:ios(config)#ssh server netconf
RP/0/RP0/CPU0:ios(config)#netconf agent tty
RP/0/RP0/CPU0:ios(config)#netconf-yang agent ssh
!
RP/0/RP0/CPU0:ios(config)#ssh timeout 120
RP/0/RP0/CPU0:ios(config)#ssh server rate-limit 600
RP/0/RP0/CPU0:ios(config)#ssh server session-limit 110
RP/0/RP0/CPU0:ios(config)#ssh server v2
RP/0/RP0/CPU0:ios(config)#ssh server vrf default
RP/0/RP0/CPU0:ios(config)#ssh server netconf vrf default
```

To configure TPA, perform the following steps:

```
RP/0/RP0/CPU0:ios(config)#tpa
RP/0/RP0/CPU0:ios(config-tpa)#linux networking
RP/0/RP0/CPU0:ios(config-tpa-vrf)#vrf default
RP/0/RP0/CPU0:ios(config-tpa-vrf)#east-west Loopback101
RP/0/RP0/CPU0:ios(config-tpa-vrf)#address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#default-route software-forwarding
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#source-hint default-route interface Loopback100
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#
```

Note TPA configuration is not required for NCS 5700 routers.

3. Reload the router (as the hw-module profile configuration is performed).
4. Check the device connection to the DDoS controller using the **ping** command.

```
RP/0/RP0/CPU0:ios#ping 10.105.237.54
Thu Jun  1 07:16:43.654 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.105.237.54 timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/4 ms
RP/0/RP0/CPU0:Router#bash
Thu Jun  1 07:16:53.024 UTC
[Router:~]$ping 10.105.237.54
PING 10.105.237.54 (10.105.237.54) 56(84) bytes of data.
64 bytes from 10.105.237.54: icmp_seq=1 ttl=63 time=1.73 ms
64 bytes from 10.105.237.54: icmp_seq=2 ttl=63 time=1.29 ms
64 bytes from 10.105.237.54: icmp_seq=3 ttl=63 time=1.27 ms
64 bytes from 10.105.237.54: icmp_seq=4 ttl=63 time=1.75 ms
^C
--- 10.105.237.54 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.270/1.510/1.751/0.230 ms
[Router:~]$
```

5. Add device details on the controller panel and ensure that all the three indicators (Deployment, Container, and Configuration) are green.

   For more information on installing the DDoS controller, see the DDoS Edge Protection Installation guide.

For more information on the DDoS Edge Protection, see Cisco Secure DDoS Edge Protection Data Sheet.

Verify DDoS Edge Protection Application Configuration

You can also verify if the DDoS controller pushes the CLI to the device using the following **show running-config** commands on the device:

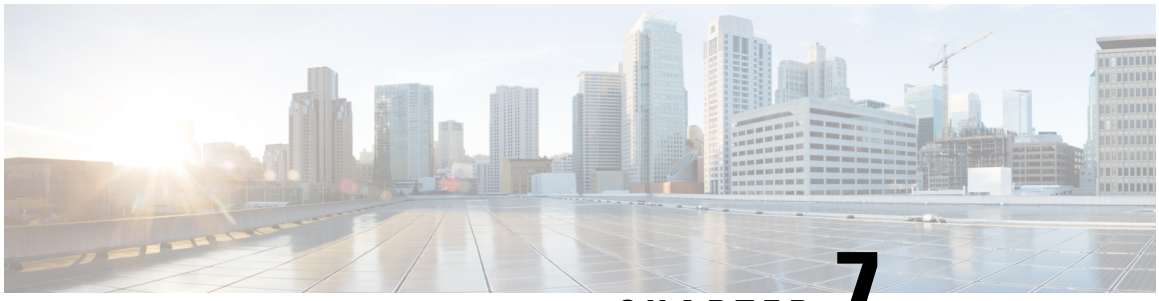
```
RP/0/RP0/CPU0:Router#show running-config appmgr
Thu Jun  1 07:33:36.741 UTC
appmgr
 application esentryd
  activate type docker source esentryd-cisco-20230431633 docker-run-opts "-p 10000:10000/tcp
 -p 5005:5005/udp --env-file /harddisk:/ENV_6478443711ac6830700d1aeb --net=host"
 !
!
```

```
RP/0/RP0/CPU0:Router#show flow monitor DetectPro_Monitor_IPV4 cache location 0/0/CPU0
Thu Nov 16 06:13:38.066 UTC
Cache summary for Flow Monitor DetectPro_Monitor_IPV4:
Cache size:                         1000000
Current entries:                          0
Flows added:                     2243884200
Flows not added:                          0
Ager Polls:                      2243884200
  - Active timeout                        0
  - Inactive timeout                      0
  - Immediate                             0
  - TCP FIN flag                          0
  - Emergency aged                        0
  - Counter wrap aged                     0
  - Total                        2243884200
Periodic export:
  - Counter wrap                          0
  - TCP FIN flag                          0
Flows exported                   2243884200


Matching entries:                         0
!
```

```
RP/0/RP0/CPU0:Router#show flow monitor DetectPro_Monitor_IPV6 cache location 0/0/CPU0
Thu Nov 16 06:13:43.734 UTC
Cache summary for Flow Monitor DetectPro_Monitor_IPV6:
Cache size:                         1000000
Current entries:                          0
Flows added:                          59971
Flows not added:                          0
Ager Polls:                           94437
  - Active timeout                    59971
  - Inactive timeout                      0
  - Immediate                             0
  - TCP FIN flag                          0
  - Emergency aged                        0
  - Counter wrap aged                     0
  - Total                             59971
Periodic export:
  - Counter wrap                          0
  - TCP FIN flag                          0
Flows exported                        59971


Matching entries:                         0
```

```
RP/0/RP0/CPU0:Router#show  flow exporter
exporter  exporter-map
```

```
RP/0/RP0/CPU0:tortin#show  flow exporter DetectPro_GPB location 0/0/CPU0
Thu Nov 16 06:13:58.059 UTC
Flow Exporter: DetectPro_GPB
Export Protocol: protobuf
Flow Exporter memory usage: 5265344
Used by flow monitors: DetectPro_Monitor_IPV4
                       DetectPro_Monitor_IPV6

Status: Disabled
Transport:   UDP
Destination: 15.1.1.2        (5005) VRF default
Source:      0.0.0.0         (54482)
Flows exported:                                  0 (0 bytes)
Flows dropped:                                   0 (0 bytes)

Templates exported:                              0 (0 bytes)
Templates dropped:                               0 (0 bytes)

Option data exported:                            0 (0 bytes)
Option data dropped:                             0 (0 bytes)

Option templates exported:                       0 (0 bytes)
Option templates dropped:                        0 (0 bytes)

Packets exported:                         20355756 (27716506821 bytes)
Packets dropped:                                 0 (0 bytes)

Total export over last interval of:
  1 hour:                                       12 pkts
                                              1879 bytes
                                                12 flows
  1 minute:                                      0 pkts
                                                 0 bytes
                                                 0 flows
  1 second:                                      0 pkts
                                                 0 bytes
                                                 0 flows

RP/0/RP0/CPU0:Router#show appmgr application-table
Thu Nov 16 06:13:58.059 UTC
Name     Type   Config State Status
-------- ------ ------------ ------------------------------------------------
esentryd Docker  Activated   Up 8 minutes
RP/0/RP0/CPU0:Router#
```



CHAPTER 7

Use Cases: Application Hosting

This chapter describes use cases for running applications on IOS XR.

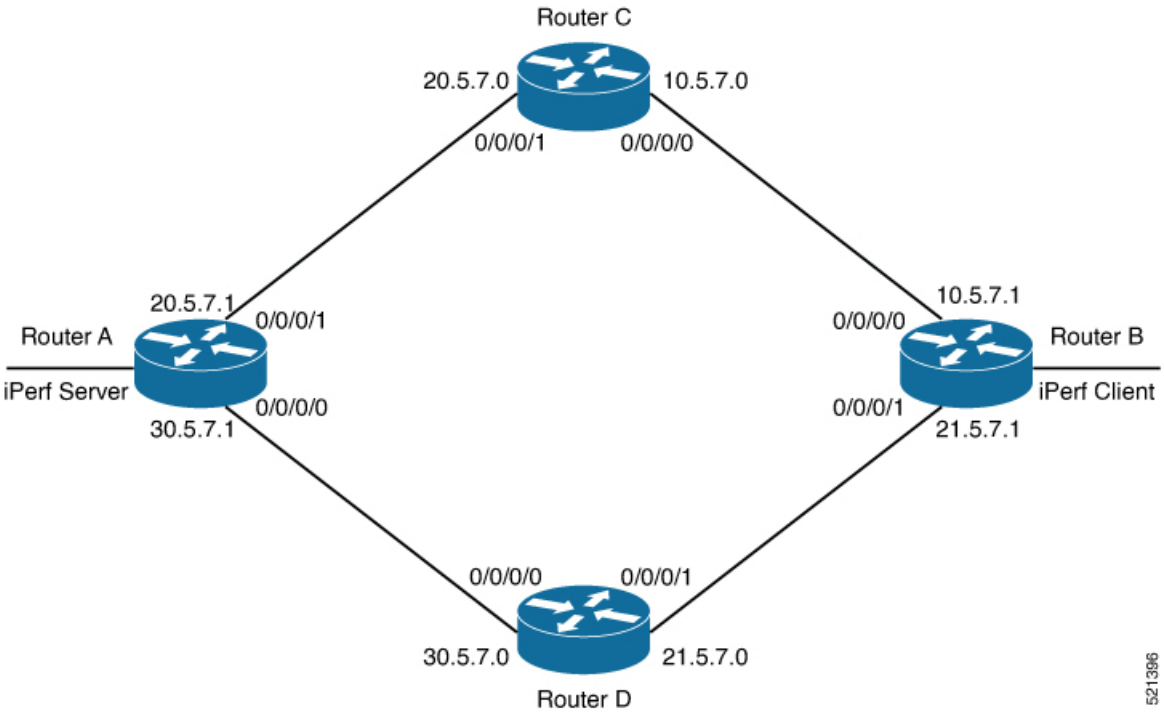
Hosting iPerf in Docker Containers to Measure Network Performance using Application Manager

Measuring the network performance is important to test the efficiency of the network. Network throughput, bandwidth, latency, and packet loss are some of the parameters used to measure the network performance. iPerf is a commonly used application for measuring network performance. The iPerf application is hosted on systems at both ends of the connection that is measured. One system is used as the server, and the other system is used as the client. At least one system must be a Cisco IOS XR router, the other system can be any other external entity like a controller or another router.

This use case illustrates the procedure for hosting the iPerf application in docker containers on two Cisco IOS XR routers, Router A and Router B to measure network performance. Router A hosts the iPerf server and Router B hosts the iPerf client.

In this usecase, we demonstrate the example of testing network bandwidth when a route update takes place. Router A hosts the iPerf Server and Router B hosts the iPerf Client. Router C and Router D are intermediate routers that allow traffic flow from Router A to Router B and vice-versa.

Figure 18: Hosting iPerf Application in Cisco IOS XR Routers



Verify Connection between Router A and Router B

The **ping** command verifies the connection between the IOS XR software on the routers, while the **bash ping** command verifies the connection between the linux kernel that hosts the IOS XR software on the routers.

Check the connection between Router A and Router B using the **ping** and **bash ping** commands.

```
Router#show ip route 30.5.7.1
Tue Dec  1 19:27:28.623 UTC

Routing entry for 30.5.7.0/31
  Known via "ospf 10", distance 110, metric 2, type intra area
  Installed Dec  1 18:09:44.525 for 01:17:44
  Routing Descriptor Blocks
    21.5.7.0, from 100.0.0.7, via FourHundredGigE0/0/0/1
      Route metric is 2
  No advertising protos.
Router#ping 30.5.7.1
Tue Dec  1 19:27:28.769 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 30.5.7.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/24/30 ms
Router#bash ping -c 5 30.5.7.1
PING 30.5.7.1 (30.5.7.1) 56(84) bytes of data.
64 bytes from 30.5.7.1: icmp_seq=1 ttl=254 time=31.9 ms
64 bytes from 30.5.7.1: icmp_seq=2 ttl=254 time=37.7 ms
64 bytes from 30.5.7.1: icmp_seq=3 ttl=254 time=30.5 ms
64 bytes from 30.5.7.1: icmp_seq=4 ttl=254 time=27.5 ms
```

```
64 bytes from 30.5.7.1: icmp_seq=5 ttl=254 time=30.3 ms

--- 30.5.7.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 27.549/31.621/37.719/3.371 ms
```

Install the iPerf Server Application

Step 1 Install the iPerf application RPM on Router A. Only the RPM file format is supported.

```
Router#appmgr package install rpm /misc/disk1/iperf-0.1.0-XR_7.3.1.x86_64.rpm

Router#show appmgr source-table
Thu Dec  3 09:57:40.808 UTC
Name            File
--------------- ------------------------------------------------------------------
iperf    iperf.tar.gz
Router#
```

Step 2 Configure the application to run as iPerf server.

```
Router#config
Thu Dec  3 09:57:54.034 UTC
Router(config)#appmgr
Router(config-appmgr)#application iperf-server-app
Router(config-application)#activate type docker source iperf docker-run-opts "--net=host" docker-run-cmd
 "iperf3 -s -d"
Router(config-application)#commit
Thu Dec  3 09:57:54.398 UTC
```

Step 3 Verify the basic details (application name and state) about the activated iPerf server application.

```
Router#show appmgr application-table
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   Up 2 seconds
Router#
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-server-app info summary
Thu Dec  3 09:58:15.569 UTC
Application: iperf-server-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Container ID: 0118f9006cde2787e9809eb7c62ad8b552925b559a689c7aaa80f80d7ce43c02
    Image: alpine1:latest
    Command: "iperf3 -s -d"
    Status: Up 7 seconds
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-server-app info detail
Thu Dec  3 09:58:26.401 UTC
Application: iperf-server-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Docker Information:
        Container ID: 0118f9006cde2787e9809eb7c62ad8b552925b559a689c7aaa80f80d7ce43c02
        Container name: iperf-server-app
        Labels:
```

```
        Image: alpine1:latest
        Command: "iperf3 -s -d"
        Created at: 2020-12-03 09:58:08 +0000 UTC
        Running for: 18 seconds ago
        Status: Up 18 seconds
        Size: 0B
        Ports:
        Mounts:
        Networks: host
        LocalVolumes: 0
Router#show appmgr application name iperf-server-app stats
Thu Dec  3 09:58:39.594 UTC
Application Stats: iperf-server-app
   CPU Percentage: 0.00%
   Memory Usage: 624KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 1
Router#
```

Step 4 Verify if the iPerf server is listening on the default port (5201) by using the netstat command inside the container.

The appmgr application exec name *app_name* docker-exec-cmd command can be used to execute any commands inside the container.

```
Router#appmgr application exec name iperf-server-app docker-exec-cmd name netstat -lnput
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:46727        0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:5201          0.0.0.0:*               LISTEN      -
udp        0      0 127.0.0.11:39552        0.0.0.0:*
Router#
```

Install the iPerf Client Application

Step 1 Install the iPerf application RPM on Router B.

```
Router#appmgr package install rpm /misc/disk1/iperf-0.1.0-XR_7.3.1.x86_64.rpm
Router#show appmgr source-table
Thu Dec  3 09:57:40.808 UTC
Name            File
--------------- ------------------------------------------------------------------
iperf      iperf.tar.gz
Router#
```

Step 2 Configure the application to run as iPerf client with a timeout (600s in this case).

```
Router#config
Thu Dec  3 09:57:54.034 UTC
Router(config)#appmgr
Router(config-appmgr)#application iperf-client-app
Router(config-application)#activate type docker source iperf docker-run-opts "--net=host" docker-run-cmd
 "iperf3 -c 30.5.7.1 -t 600"
Router(config-application)#commit
Thu Dec  3 09:57:54.398 UTC
```

Note Hosting the iPerf client application on Router B by providing the iPerf server physical interface IP address (30.5.7.1) establishes communication between Router B and Router A.

Step 3 Verify the basic details (application name and state) about the activated iPerf client application.

```
Router#show appmgr application-table
Thu Dec  3 09:59:47.628 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-client-app     Docker    Activated   Up 2 seconds
Router#
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-client-app info summary
Thu Dec  3 09:59:54.534 UTC
Application: iperf-client-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Container ID: 40e1730a97666b2b44c8c9313b94b0138925c9198ae63244ff3bd386132d9c9c
    Image: alpine1:latest
    Command: "iperf3 -c 30.5.7.1 -t 600"
    Status: Up 9 seconds
Router#show appmgr application name iperf-client-app info detail
Application: iperf-client-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Docker Information:
        Container ID: 40e1730a97666b2b44c8c9313b94b0138925c9198ae63244ff3bd386132d9c9c
        Container name: iperf-client-app
        Labels:
        Image: alpine1:latest
        Command: "iperf3 -c 30.5.7.1 -t 600"
        Created at: 2020-12-03 09:59:45 +0000 UTC
        Running for: 20 seconds ago
        Status: Up 20 seconds
        Size: 0B
        Ports:
        Mounts:
        Networks: host
        LocalVolumes: 0
Router#show appmgr application name iperf-client-app stats
Thu Dec  3 10:00:18.079 UTC
Application Stats: iperf-client-app
   CPU Percentage: 0.11%
   Memory Usage: 720KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 1
Router#
```

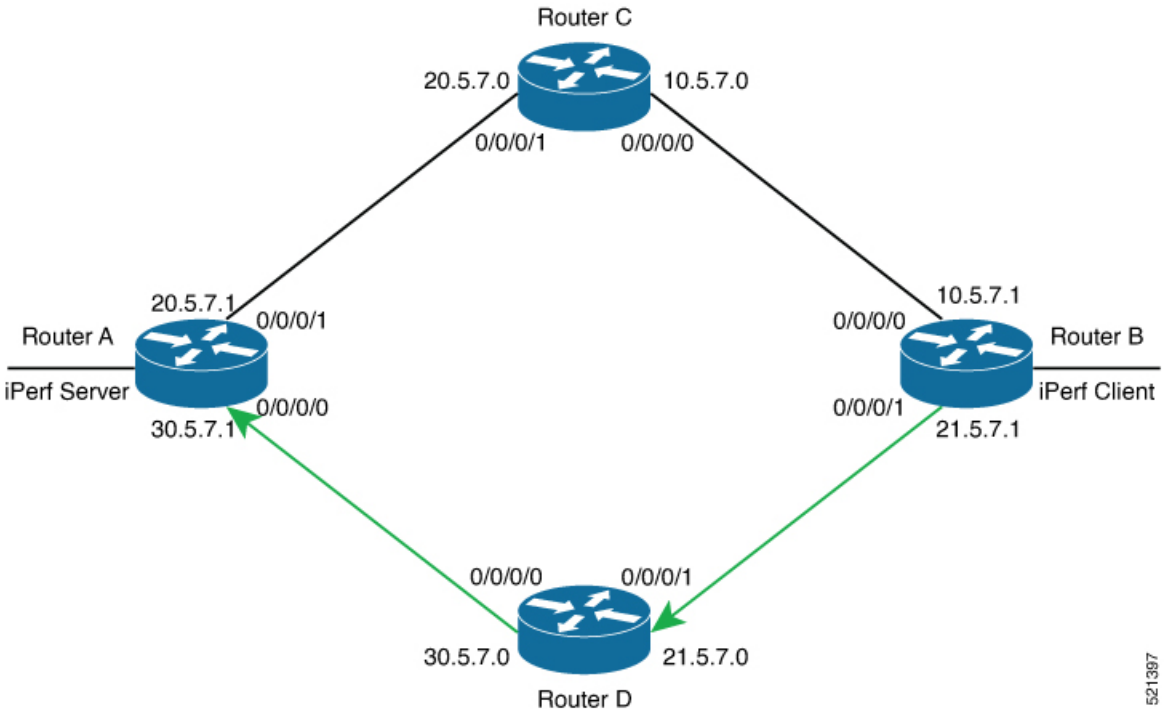
Verify Connection between the iPerf Server and iPerf Client Applications

Verify whether the connection is established between iPerf server and iPerf clients by executing the **bash netstat -anput** command on Router A. When the iPerf client is up and running, the entry in the **State** field displays "ESTABLISHED".

```
Router#bash netstat -anput
Thu Dec  3 10:00:33.535 UTC
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:646             0.0.0.0:*               LISTEN      8585/mpls_ldp
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      8567/ssh_server
tcp        0      0 0.0.0.0:830             0.0.0.0:*               LISTEN      8567/ssh_server
tcp6       0      0 :::5201                 :::*                    LISTEN      20829/iperf3
tcp6       0      0 :::22                   :::*                    LISTEN      8567/ssh_server
tcp6       0      0 :::830                  :::*                    LISTEN      8567/ssh_server
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65322         ESTABLISHED 20829/iperf3
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65302         ESTABLISHED 20829/iperf3
udp        0      0 0.0.0.0:646             0.0.0.0:*                           8585/mpls_ldp
udp        0      0 0.0.0.0:3232            0.0.0.0:*                           6833/pim
udp        0      0 0.0.0.0:3503            0.0.0.0:*                           10762/lspv_server
udp        0      0 0.0.0.0:68              0.0.0.0:*                           10704/xr_dhcpcd
udp        0      0 0.0.0.0:496             0.0.0.0:*                           6833/pim
udp6       0      0 :::3503                 :::*                                10762/lspv_server
```

Measure Network Performance

Step 1 Verify the traffic route from Router B to Router A using the **show ip route** command, on Router B.




```
Router#show ip route 30.5.7.1
Thu Dec  3 10:08:01.859 UTC

Routing entry for 30.5.7.0/31
  Known via "ospf 10", distance 110, metric 2, type intra area
  Installed Dec  3 04:49:22.281 for 05:18:39
```

```
  Routing Descriptor Blocks
    21.5.7.0, from 100.0.0.7, via FourHundredGigE0/0/0/1
      Route metric is 2
  No advertising protos.
Router#
```

Step 2 Check the network performance between iPerf client and iPerf server (on Router B and Router A).

You can view the network monitoring parameters by executing the **show appmgr application name iperf-client-app logs** command, on Router B that hosts the iPerf client.

```
Router#show appmgr application name iperf-client-app logs
Tue Dec 1 12:50:27.862 UTC
Connecting to host 30.5.7.1, port 5201
[ 4] local 100.0.0.9 port 61384 connected to 30.5.7.1 port 5201
[ ID] Interval          Transfer     Bandwidth Retr     Cwnd
[ 4] 0.00-1.00 sec 1.05 MBytes     8.82 Mbits/sec 0     80.6 KBytes
[ 4] 1.00-2.00 sec 1.26 MBytes     10.6 Mbits/sec 0     136 KBytes
[ 4] 2.00-3.00 sec 1.18 MBytes     9.90 Mbits/sec 0     191 KBytes
[ 4] 3.00-4.00 sec 1.24 MBytes     10.4 Mbits/sec 0     246 KBytes
[ 4] 4.00-5.00 sec 1.18 MBytes     9.90 Mbits/sec 0     301 KBytes
[ 4] 5.00-6.00 sec 1.37 MBytes     11.5 Mbits/sec 0     362 KBytes
[ 4] 6.00-7.00 sec 1.37 MBytes     11.5 Mbits/sec 0     423 KBytes
[ 4] 7.00-8.00 sec 1.43 MBytes     12.0 Mbits/sec 0     486 KBytes
[ 4] 8.00-9.00 sec 1.30 MBytes     11.0 Mbits/sec 0     547 KBytes
[ 4] 9.00-10.00 sec 1.43 MBytes     12.0 Mbits/sec 0    611 KBytes
[ 4] 10.00-11.00 sec 1.62 MBytes     13.6 Mbits/sec 0   707 KBytes
[ 4] 11.00-12.00 sec 1.62 MBytes     13.6 Mbits/sec 0    875 KBytes
[ 4] 12.00-13.00 sec 1.93 MBytes     16.2 Mbits/sec 0    1.07 MBytes
[ 4] 13.00-14.00 sec 1.68 MBytes     14.1 Mbits/sec 0    1.29 MBytes
[ 4] 14.00-15.00 sec 1.06 MBytes     8.86 Mbits/sec 0    1.56 MBytes
[ 4] 15.00-16.00 sec 891 KBytes     7.30 Mbits/sec 0     1.83 MBytes
[ 4] 16.00-17.00 sec 970 KBytes     7.95 Mbits/sec 0     2.12 MBytes
[ 4] 17.00-18.00 sec 1.24 MBytes     10.4 Mbits/sec 0    2.58 MBytes
[ 4] 18.00-19.00 sec 885 KBytes     7.24 Mbits/sec 0     2.65 MBytes
[ 4] 19.00-20.00 sec 1.55 MBytes     13.0 Mbits/sec 0    3.10 MBytes
[ 4] 20.00-21.00 sec 820 KBytes     6.71 Mbits/sec 0     3.10 MBytes
[ 4] 21.00-22.00 sec 1.72 MBytes     14.4 Mbits/sec 6     2.42 MBytes
[ 4] 22.00-23.00 sec 0.00 Bytes     0.00 bits/sec 5       2.30 MBytes
[ 4] 23.00-24.00 sec 256 KBytes     2.10 Mbits/sec 0     1.35 MBytes
[ 4] 24.00-25.00 sec 1.56 MBytes     13.1 Mbits/sec 237   1.83 MBytes
[ 4] 25.00-26.00 sec 1.90 MBytes     15.9 Mbits/sec 0     2.17 MBytes
[ 4] 26.00-27.00 sec 382 KBytes     3.12 Mbits/sec 61     1.95 MBytes
[ 4] 27.00-28.00 sec 0.00 Bytes     0.00 bits/sec 0     1.39 MBytes
[ 4] 28.00-29.00 sec 3.35 MBytes     28.1 Mbits/sec 0     1.52 MBytes
[ 4] 29.00-30.00 sec 954 KBytes     7.82 Mbits/sec 0     1.58 MBytes
[ 4] 30.00-31.00 sec 1018 KBytes     8.34 Mbits/sec 0     1.64 MBytes
[ 4] 31.00-32.00 sec 1.24 MBytes     10.4 Mbits/sec 0     1.71 MBytes
[ 4] 32.00-33.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.76 MBytes
[ 4] 33.00-34.00 sec 1.61 MBytes     13.5 Mbits/sec 0     1.80 MBytes
[ 4] 34.00-35.00 sec 1.46 MBytes     12.2 Mbits/sec 0     1.82 MBytes
[ 4] 35.00-36.00 sec 1.18 MBytes     9.89 Mbits/sec 0     1.83 MBytes
[ 4] 36.00-37.00 sec 1.36 MBytes     11.4 Mbits/sec 0     1.84 MBytes
[ 4] 37.00-38.00 sec 1.36 MBytes     11.4 Mbits/sec 0     1.84 MBytes
[ 4] 38.00-39.00 sec 1.24 MBytes     10.4 Mbits/sec 0     1.84 MBytes
[ 4] 39.00-40.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.85 MBytes
[ 4] 40.00-41.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.86 MBytes
[ 4] 41.00-42.00 sec 1.40 MBytes     11.8 Mbits/sec 0     1.88 MBytes
[ 4] 42.00-43.00 sec 1.12 MBytes     9.37 Mbits/sec 0     1.91 MBytes
[ 4] 43.00-44.00 sec 1.12 MBytes     9.40 Mbits/sec 0     1.96 MBytes
[ 4] 44.00-45.00 sec 1.20 MBytes     10.1 Mbits/sec 0     2.02 MBytes
[ 4] 45.00-46.00 sec 1.27 MBytes     10.7 Mbits/sec 0     2.11 MBytes
[ 4] 46.00-47.00 sec 1.30 MBytes     10.9 Mbits/sec 0     2.22 MBytes
```

```
[ 4] 47.00-48.00 sec 1.25 MBytes     10.5 Mbits/sec 0     2.36 MBytes
[ 4] 48.00-49.00 sec 1.43 MBytes     12.0 Mbits/sec 0     2.53 MBytes
```

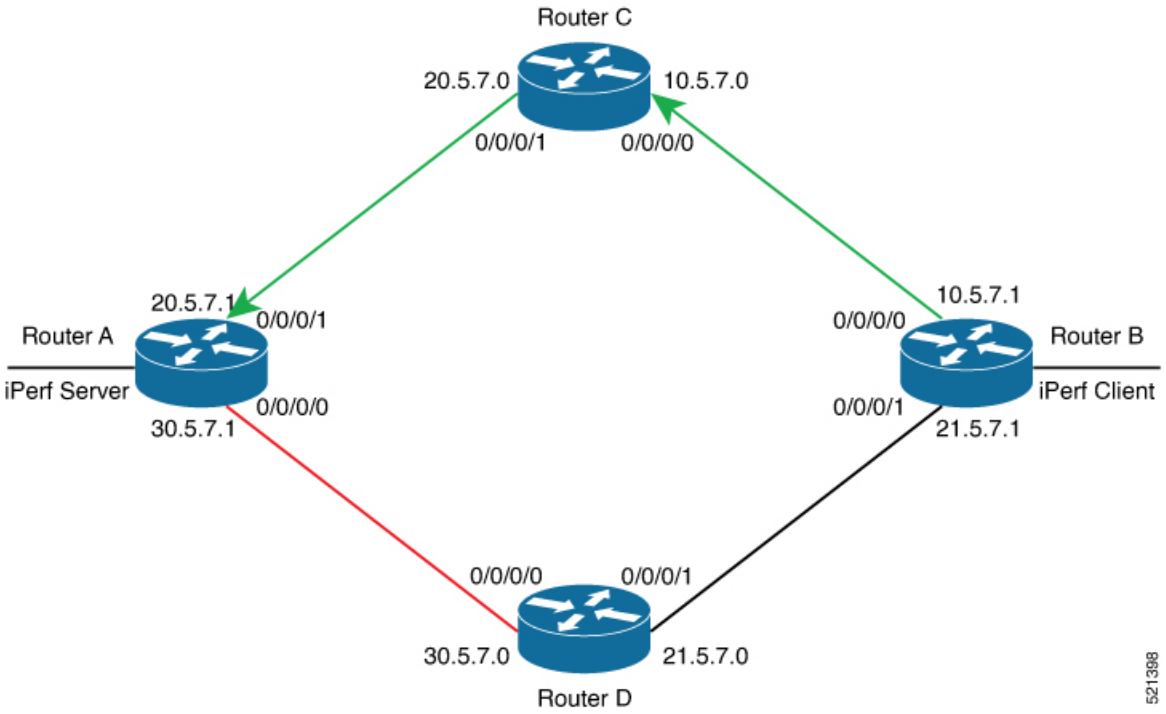
Step 3 Bring down the interface on Router D using the **shut** command to trigger a route update.

```
Router(config)#interface FourhundredGig0/0/0/0
Router(config-if)#shut
Router(config-if)#commit
```

Note Because of the interface shutdown, the route to 30.5.7.1 needs to be updated and hence momentarily there will be no route to this address.

Step 4 During the route update, check the network performance by executing the **show appmgr application name** *app_name* **logs** command.

You will notice that the entries in the **Bandwidth** field is Zero for a short duration, when the new route is installed.




```
Router#show appmgr application name iperf-client-app logs
Tue Dec 1 12:59:40.349 UTC
Connecting to host 30.5.7.1, port 5201
[ 4] local 100.0.0.9 port 61384 connected to 30.5.7.1 port 5201
15
[ ID] Interval          Transfer Bandwidth     Retr     Cwnd
[ 4] 0.00-1.00 sec 1.05 MBytes 8.82 Mbits/sec 0     80.6 KBytes
[ 4] 1.00-2.00 sec 1.26 MBytes 10.6 Mbits/sec 0     136 KBytes
[ 4] 2.00-3.00 sec 1.18 MBytes 9.90 Mbits/sec 0     191 KBytes
[ 4] 3.00-4.00 sec 1.24 MBytes 10.4 Mbits/sec 0     246 KBytes
[ 4] 4.00-5.00 sec 1.18 MBytes 9.90 Mbits/sec 0     301 KBytes
[ 4] 5.00-6.00 sec 1.37 MBytes 11.5 Mbits/sec 0     362 KBytes
[ 4] 6.00-7.00 sec 1.37 MBytes 11.5 Mbits/sec 0     423 KBytes
[ 4] 7.00-8.00 sec 1.43 MBytes 12.0 Mbits/sec 0     486 KBytes
[ 4] 8.00-9.00 sec 1.30 MBytes 11.0 Mbits/sec 0     547 KBytes
[ 4] 9.00-10.00 sec 1.43 MBytes 12.0 Mbits/sec 0     611 KBytes
[ 4] 10.00-11.00 sec 1.62 MBytes 13.6 Mbits/sec 0     707 KBytes
[ 4] 11.00-12.00 sec 1.62 MBytes 13.6 Mbits/sec 0     875 KBytes
```

```
[ 4] 12.00-13.00 sec 1.93 MBytes 16.2 Mbits/sec 0     1.07 MBytes
[ 4] 13.00-14.00 sec 1.68 MBytes 14.1 Mbits/sec 0     1.29 MBytes
[ 4] 14.00-15.00 sec 1.06 MBytes 8.86 Mbits/sec 0     1.56 MBytes
[ 4] 15.00-16.00 sec 891 KBytes 7.30 Mbits/sec 0     1.83 MBytes
[ 4] 16.00-17.00 sec 970 KBytes 7.95 Mbits/sec 0     2.12 MBytes
[ 4] 17.00-18.00 sec 1.24 MBytes 10.4 Mbits/sec 0     2.58 MBytes
[ 4] 18.00-19.00 sec 885 KBytes 7.24 Mbits/sec 0     2.65 MBytes
[ 4] 19.00-20.00 sec 1.55 MBytes 13.0 Mbits/sec 0     3.10 MBytes
[ 4] 20.00-21.00 sec 820 KBytes 6.71 Mbits/sec 0     3.10 MBytes
[ 4] 21.00-22.00 sec 1.72 MBytes 14.4 Mbits/sec 6     2.42 MBytes
[ 4] 22.00-23.00 sec 0.00 Bytes 0.00 bits/sec 5     2.30 MBytes
[ 4] 23.00-24.00 sec 256 KBytes 2.10 Mbits/sec 0     1.35 MBytes
[ 4] 24.00-25.00 sec 1.56 MBytes 13.1 Mbits/sec 237     1.83 MBytes
[ 4] 25.00-26.00 sec 1.90 MBytes 15.9 Mbits/sec 0     2.17 MBytes
[ 4] 26.00-27.00 sec 382 KBytes 3.12 Mbits/sec 61     1.95 MBytes
[ 4] 27.00-28.00 sec 0.00 Bytes 0.00 bits/sec 0     1.39 MBytes
[ 4] 28.00-29.00 sec 3.35 MBytes 28.1 Mbits/sec 0     1.52 MBytes
[ 4] 29.00-30.00 sec 954 KBytes 7.82 Mbits/sec 0     1.58 MBytes
[ 4] 30.00-31.00 sec 1018 KBytes 8.34 Mbits/sec 0     1.64 MBytes
[ 4] 31.00-32.00 sec 1.24 MBytes 10.4 Mbits/sec 0     1.71 MBytes
[ 4] 32.00-33.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.76 MBytes
[ 4] 33.00-34.00 sec 1.61 MBytes 13.5 Mbits/sec 0     1.80 MBytes
[ 4] 34.00-35.00 sec 1.46 MBytes 12.2 Mbits/sec 0     1.82 MBytes
[ 4] 35.00-36.00 sec 1.18 MBytes 9.89 Mbits/sec 0     1.83 MBytes
[ 4] 36.00-37.00 sec 1.36 MBytes 11.4 Mbits/sec 0     1.84 MBytes
[ 4] 37.00-38.00 sec 1.36 MBytes 11.4 Mbits/sec 0     1.84 MBytes
[ 4] 38.00-39.00 sec 1.24 MBytes 10.4 Mbits/sec 0     1.84 MBytes
[ 4] 39.00-40.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.85 MBytes
[ 4] 40.00-41.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.86 MBytes
[ 4] 41.00-42.00 sec 1.40 MBytes 11.8 Mbits/sec 0     1.88 MBytes
[ 4] 42.00-43.00 sec 1.12 MBytes 9.37 Mbits/sec 0     1.91 MBytes
[ 4] 43.00-44.00 sec 1.12 MBytes 9.40 Mbits/sec 0     1.96 MBytes
[ 4] 44.00-45.00 sec 1.20 MBytes 10.1 Mbits/sec 0     2.02 MBytes
[ 4] 45.00-46.00 sec 1.27 MBytes 10.7 Mbits/sec 0     2.11 MBytes
[ 4] 46.00-47.00 sec 1.30 MBytes 10.9 Mbits/sec 0     2.22 MBytes
[ 4] 95.00-96.00 sec 1.48 MBytes 12.4 Mbits/sec 0     1.82 MBytes
[ 4] 96.00-97.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.83 MBytes
[ 4] 97.00-98.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.83 MBytes
[ 4] 98.00-99.00 sec 1.49 MBytes 12.5 Mbits/sec 0     1.84 MBytes
[ 4] 99.00-100.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.86 MBytes
[ 4] 100.00-101.00 sec 1.21 MBytes 10.2 Mbits/sec 0     1.89 MBytes
[ 4] 101.00-102.00 sec 1.34 MBytes 11.2 Mbits/sec 0     1.94 MBytes
[ 4] 102.00-103.00 sec 1.25 MBytes 10.5 Mbits/sec 0     2.01 MBytes
[ 4] 103.00-104.00 sec 1.30 MBytes 10.9 Mbits/sec 0     2.09 MBytes
[ 4] 104.00-105.00 sec 1.25 MBytes 10.5 Mbits/sec 0     2.17 MBytes
[ 4] 105.00-106.00 sec 1.39 MBytes 11.6 Mbits/sec 0     2.33 MBytes
[ 4] 106.00-107.00 sec 1.01 MBytes 8.47 Mbits/sec 0     2.46 MBytes
[ 4] 107.00-108.00 sec 526 KBytes 4.31 Mbits/sec 0     2.54 MBytes
[ 4] 108.00-109.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 109.00-110.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 110.00-111.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 111.00-112.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 112.00-113.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 113.00-114.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 114.00-115.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 115.00-116.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 116.00-117.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 117.00-118.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 118.00-119.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 119.00-120.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 120.00-121.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 121.00-122.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 122.00-123.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 123.00-124.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
```

```
[ 4] 124.00-125.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 125.00-126.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 126.00-127.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 127.00-128.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 128.00-129.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 129.00-130.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 130.00-131.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 131.00-132.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 132.00-133.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 133.00-134.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 134.00-135.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 135.00-136.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 136.00-137.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 137.00-138.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 138.00-139.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 139.00-140.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 140.00-141.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 141.00-142.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 142.00-143.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 143.00-144.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 144.00-145.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 145.00-146.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 146.00-147.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 147.00-148.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 148.00-149.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 149.00-150.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 150.00-151.00 sec 700 KBytes 5.73 Mbits/sec 847    600 KBytes
[ 4] 151.00-152.00 sec 954 KBytes 7.82 Mbits/sec 993    1.32 MBytes
[ 4] 152.00-153.00 sec 509 KBytes 4.17 Mbits/sec 0     1.79 MBytes
[ 4] 153.00-154.00 sec 1.08 MBytes 9.07 Mbits/sec 0     1.85 MBytes
[ 4] 154.00-155.00 sec 1.38 MBytes 11.6 Mbits/sec 0     1.90 MBytes
[ 4] 155.00-156.00 sec 1.55 MBytes 13.0 Mbits/sec 0     1.98 MBytes
[ 4] 156.00-157.00 sec 1.16 MBytes 9.71 Mbits/sec 0     2.04 MBytes
[ 4] 157.00-158.00 sec 1.21 MBytes 10.2 Mbits/sec 0     2.10 MBytes
[ 4] 158.00-159.00 sec 1.26 MBytes 10.6 Mbits/sec 0     2.17 MBytes
[ 4] 159.00-160.00 sec 1.14 MBytes 9.56 Mbits/sec 0     2.23 MBytes
[ 4] 160.00-161.00 sec 1.29 MBytes 10.8 Mbits/sec 0     2.27 MBytes
[ 4] 161.00-162.00 sec 1.24 MBytes 10.4 Mbits/sec 0     2.34 MBytes
[ 4] 162.00-163.00 sec 1.42 MBytes 11.9 Mbits/sec 0     2.41 MBytes
[ 4] 163.00-164.00 sec 1.11 MBytes 9.34 Mbits/sec 0     2.46 MBytes
[ 4] 164.00-165.00 sec 1.39 MBytes 11.7 Mbits/sec 0     2.56 MBytes
[ 4] 165.00-166.00 sec 995 KBytes 8.16 Mbits/sec 0     2.69 MBytes
[ 4] 166.00-167.00 sec 1.88 MBytes 15.7 Mbits/sec 0     2.94 MBytes
[ 4] 167.00-168.02 sec 950 KBytes 7.69 Mbits/sec 0     3.12 MBytes
[ 4] 168.02-169.00 sec 1.79 MBytes 15.2 Mbits/sec 0     3.12 MBytes
[ 4] 169.00-170.01 sec 1.27 MBytes 10.6 Mbits/sec 0     3.12 MBytes
[ 4] 170.01-171.00 sec 1.25 MBytes 10.5 Mbits/sec 23     1.60 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-600.00 sec   704 MBytes  9.84 Mbits/sec  12069   sender
[  4]   0.00-600.00 sec   702 MBytes  9.82 Mbits/sec         receiver

iperf Done.

<!-On Router A!>
Router#show appmgr application name iperf-server-app stats
Thu Dec  3 11:45:47.790 UTC
Application Stats: iperf-server-app
   CPU Percentage: 0.00%
   Memory Usage: 816KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
```

```
   PIDs: 1
<!-On Router B!>
Router#show appmgr application name iperf-client-app stats
Thu Dec  3 11:45:59.418 UTC
Application Stats: iperf-client-app
   CPU Percentage: 0.00%
   Memory Usage: 0B / 0B
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 0
```

Stop iPerf Applications

Stop the iPerf applications on Router A and Router B using the **appmgr application stop name** *app_name* command. The **application stop** command can only be used for applications that are registered, activated, and are currently running. The **application stop** command stops only the application and does not clean up the resources used by the application.

You can verify the status of the application using the **show appmgr application-table** command. The **Status** is displayed as **Exited** if the application has been stopped successfully.

```
Router#appmgr application stop name iperf-server-app
Mon Nov 30 13:38:36.202 UTC
Router#show appmgr application-table
Mon Nov 30 13:38:36.999 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   Exited (1) Less than a se
Router#
```

Start iPerf Applications

Start or restart an application that has been stopped (and not deactivated) using the **appmgr application start name** *app_name* command.

```
Router#appmgr application start name iperf-server-app
Tue Dec 1 13:06:21.996 UTC
Router#show appmgr application-table
Mon Nov 30 13:38:36.999 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   UP(1) Less than a second
Router#
```

Deactivate iPerf Applications

Step 1 Deactivate the iPerf applications using the **no appmgr application** *app_name* command. You deactivate the installed application when you want to release all resources used by the application.

```
Router#config
Router(config)#no appmgr application iperf-server-app
Router(config)#commit
```

Step 2 Verify the status of the application by using the **show appmgr application-table** *app_name* **stats** command.

```
Router#show appmgr application-table
Mon Nov 30 13:39:51.197 UTC
Router#
```

Note You can activate a deactivated application using the **appmgr application** *app_name* **activate type docker source** *source_name* command.

Uninstall iPerf Applications

Uninstall the applications using the **appmgr package uninstall package** *package_name* command.

After the application is successfully uninstalled, executing the **show appmgr source-table** command displays no result.

```
Router#appmgr package uninstall package iperf
table
Mon Nov 30 13:41:05.155 UTC
Router#show appmgr source-table
Mon Nov 30 13:41:05.936 UTC
Router#
```

CPU-Based Packet Generator

Table 17: Feature History Table

Feature Name	Release Information	Feature Description
CPU-Based Packet Generator on NCS 5700 fixed port routers	Release 24.2.11	Introduced in this release on: NCS 5700 fixed port routers This feature support is now extended to NCS 5700 fixed port routers.

Feature Name	Release Information	Feature Description
CPU-Based Packet Generator	Release 24.2.1	Introduced in this release on: NCS 5500 fixed port routers; NCS 5500 modular routers(NCS 5500 line cards; NCS 5700 line cards [Mode: Compatibility; Native]) You can now use a CPU-based packet generator for IOS-XR routers to simplify the diagnostic process for routers experiencing problems. This tool allows you to generate a wide range of traffic streams directly within the production environment without physically isolating the routers and moving them to a lab setup. This tool is beneficial in environments that use routers from different vendors or different models from the same vendor. The feature introduces the CLI Options command with different options to generate different types of packets.

Need for CPU-Based Packet Generator

Diagnosing network problems in production environments, such as traffic drops and mis-forwarding issues, is crucial for network management. Traditionally, routers are physically isolated for debugging, requiring moving equipment into lab environments with traffic generators.The CPU-Based Packet Generator can be used in the production environment, eliminating the need to isolate the routers to a lab environment for troubleshooting purposes.

Benefits of CPU-Based Packet Generator

- Versatile Traffic Crafting: Create complex nested packets, such as IPinIPinIPinIP, to test and diagnose a variety of scenarios.
- In-Production Diagnosis: Directly diagnose routers in a problem state without disrupting the network setup.

Restrictions of CPU-Based Packet Generator

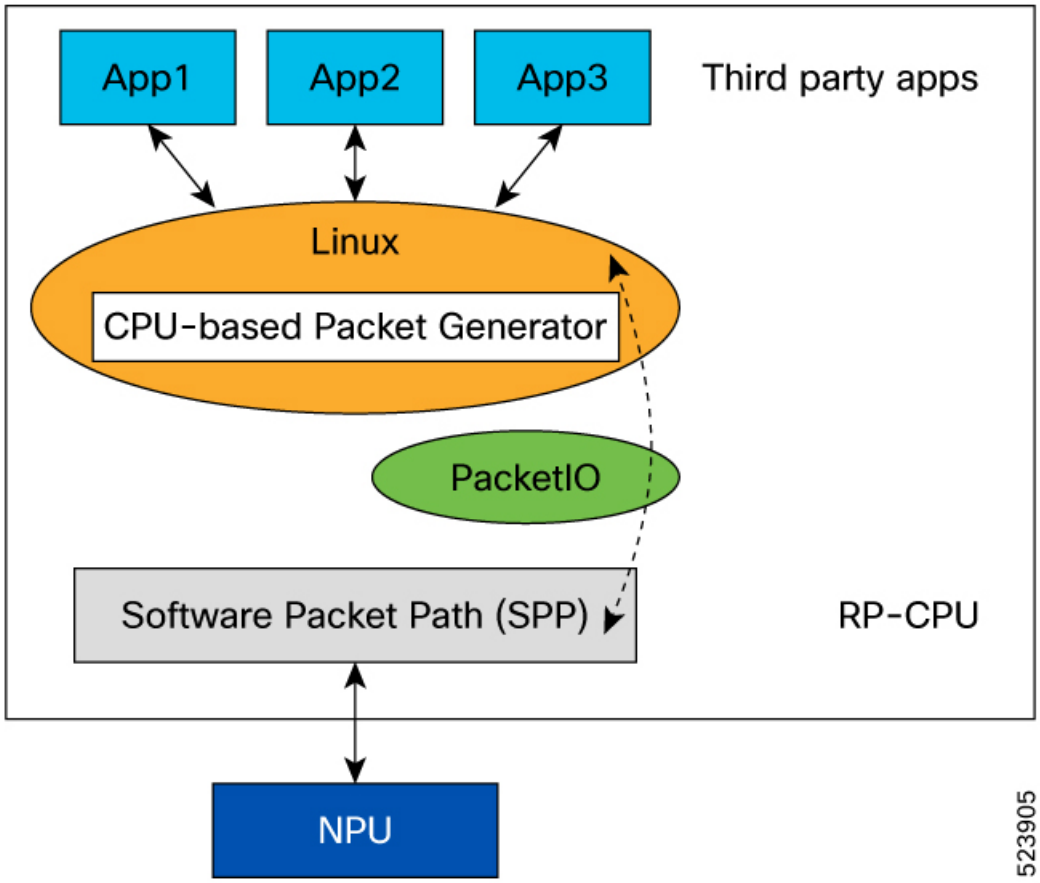
- CPU-based packet generators are not optimized for high-speed packet processing; therefore, they may not match the performance of NPU-based packet generators.
- CPU-based packet generators can potentially introduce higher CPU loads during operation, which may affect the router performance.

- The probe packet rate is 80 kpps.

Topology of CPU-Based Packet Generator

The following diagram depicts the software architecture of CPU-based packet generator.

Figure 19: Architecture of CPU-Based Packet Generator



The Cisco IOS-XR PacketIO serves as a host for third-party applications on the XR platform, with PacketIO infrastructure facilitating packet transport and interactions between Linux and XR environments. Leveraging this existing infrastructure, the CPU-based packet generator is implemented as a Linux application and packaged within the supported XR platform base image, ensuring seamless distribution.

The Linux infrastructure maintains a database of all XR interfaces including bundles. The CPU-based packet generator is used to send a specific packet type over a chosen interface.

Capabilities of CPU-based Packet Generator

- **Support different packet types:** The CPU-based packet generator supports various packet types, including:
  - ARP

- TCP
- UDP
- GRE
- MPLS
- IPinIP
- ICMPv4
- ICMPv6

- **Corrupt or error packet generation:** There are times when routers receive packets that are either corrupted or contain errors for various reasons. To identify and troubleshoot these issues, it becomes necessary to generate similar packets that can be used for debugging purposes. The CPU-based packet generator can create these packets and aid debugging.

  Examples include:

  - IPv4 packet with TTL 0
  - IPv4 packet with wrong checksum
  - IPv4 packet with mismatch between IP option length field and the IP header

How to Use CPU-based Packet Generator?

You can use CPU-based packet generator using:

- **CLI:** Use the **packetgen** command with different options to run the tool from XR bash environment. As the XR interfaces show up as Linux interfaces in bash environment, you can directly use the XR interface names.
- **pcap file:** Use an already captured pcap file in production routers and replay it.

  packetgen -i interface_name -pcap pcap_file

CLI Options

The following table outlines the different options available for the **packetgen** command.

Table 18: Packetgen CLI Options

Option	Description
-accounting	Turn on accounting for packets. Only works if packets come back to the packet generator.
-arp-destination-hw-address string	ARP target hardware address (default: uses interface MAC address)
-arp-destination-ip-address string	ARP target IP address (default: 127.0.0.1 or ::1)

Option	Description
-arp-operation uint	ARP operation (1: request, 2: reply , 3: rarp)
-arp-source-hw-address string	ARP sender hardware address (default : uses interface MAC)
-arp-source-ip-address string	ARP sender IP address (default: uses interface IP)
-burst int	Number of packets to be injected at a time. To be used in conjunction with -sleep.
-count int	Number of packets to be generated.
-data-type string	constant, incrementing, random (default: no payload)
-ethernet-dmac string	Destination MAC address (default: ff:ff:ff:ff:ff:ff)
-ethernet-smac string	Source MAC address (default: use interface MAC address)
-file string	Write packets to file
-gre	Enable GRE
-gre-checksum-present	Enable GRE checksum present bit
-gre-key-present	Enable GRE key present bit
-gre-over-mpls	Enable GRE over MPLS
-gre-protocol uint	Set the protocol type of the GRE payload (default: 0x0800 (IP)
-gre-seq-present	Enable GRE sequence number present bit
-gre-version uint	Set the GRE version number (default 0)
-header string	Custom header for all packets
-hex	Print hex dump of packets
-i string	Interface name for packet injection
-icmp-code uint	ICMP code (default: 0)
-icmp-type uint	ICMP type (default: 0)
-inc-dmac	Increment destination MAC
-inc-smac	Increment source mac
-inner-ethernet-dmac string	Inner Ethernet destination MAC address (default: ff:ff:ff:ff:ff:ff)
-inner-ethernet-smac string	Inner Ethernet source MAC address (default: ff:ff:ff:ff:ff:ff)

Option	Description
-inner-ip-checksum uint	Inner IP checksum (default: compute checksum automatically)
-inner-ip-dont-fragment uint	Set inner IP Don't Fragment flag as 1
-inner-ip-dst string	Inner destination IP address (default: 127.0.0.1 or ::1)
-inner-ip-flow-label uint	Inner IPv6 Flow Label value (default: 0)
-inner-ip-frag-offset uint	Inner IP fragment offset in units of 64-bits (e.g. 1 = 64 bits)
-inner-ip-protocol string	Inner IP protocol . Supports protocol text (TCP, UDP) and code (63 for TCP) (default: TCP)
-inner-ip-src string	Inner source IP address (default: 127.0.0.1 or ::1)
-inner-ip-tos uint	Inner IP Type Of Service (TOS) value (default: 0)
-inner-ip-traffic-class uint	ip-traffic-class (traffic-class) value (default: 0)
-inner-ip-ttl uint	Inner IP time to live (ttl). (Default ttl = 64
-inner-ip-version int	Inner IP version (default: 4)
-inner-vlan-id uint	Inner VLAN id (default: 0)
-inner-vlan-tpid uint	Inner VLAN ethernet type (default: 33024 :Dot1Q)
-inner-vlan-vpri uint	Inner VLANpriority (default: 0
-ip-checksum string	IP checksum (default: compute checksum automatically)
-ip-dont-fragment string	Set IP flag -ip-dont-fragment 0 -> 000 Nothing set -ip-dont-fragment 1 -> 001 More Fragments -ip-dont-fragment 2 -> 010 Dont Fragment -ip-dont-fragment 4 -> 100 set reserved bit
-ip-dst string	Destination IP address (default: 127.0.0.1 or ::1)
-ip-flow-label string	IPv6 Flow Label value (default: 0)
-ip-frag-offset string	Fragment offset in units of 64-bits (1 = 64 bits)
-ip-protocol string	IP protocol. Supports protocol text (TCP, UDP, GRE, VXLAN, ICMP, NDP) and code (63 for TCP) (default: TCP)
-ip-src string	Source IP address (default: use interface ip)
-ip-tos string	IP Type Of Service value (default: 0)
-ip-traffic-class string	IP traffic class (traffic-class) value (default: 0)

Option	Description
-ip-ttl string	IP time to live (ttl). (Default ttl = 64
-ip-version string	IP version should always be set for accurate IP packet creation, ip version (default: 4).
-mpls-exp string	Comma separated MPLS EXP (Experimental) value (default: 0)
-mpls-label string	Comma separated list of Multiprotocol Label Switching (MPLS) labels to be added to the packet. Specified from top to bottom
-mpls-ttl string	Comma separated MPLS TTL (Time To Live) value (default: 64)
-ndp string	Specify the neighbor discovery protocol: nbr-solicit, nbr-advt
-ndp-target-address string	NDP target address (default: for advertisement source IP, for solicitation destination IP
-pcap string	File to replay pcap
-progress	Display a progress bar
-seed int	Seed for pseudo random payload generator
-size int	Size of payload
-sleep string	Time duration to sleep during each burst. To be used together with -burst.
-stdout	Print packets to stdout
-tcp-dport int	TCP destination port (default: 40000)
-tcp-flags string	Set TCP control flags: • U (Urgent): Indicates that the data should be processed urgently. • A (Acknowledgement): Acknowledges the receipt of data. • P (Push): Instructs the sender to push the data to the receiving application immediately. • R (Reset): Resets the connection. • S (Synchronize): Synchronizes sequence numbers to initiate a connection. • F (Finish): Indicates the sender has finished sending data and wants to terminate the connection.
-tcp-sport int	TCP source port (default: 40000)
-udp-dport int	UDP destination port (default: 40000)
-udp-sport int	UDP source port (default: 40000)
-vlan-id uint	VLAN id (default: 0)

Option	Description
-vlan-tpid uint	VLAN ethernet type (default: 33024 :Dot1Q)
-vlan-vpri uint	VLAN priority (default: 0
-vxlan-udp-dport int	UDP destination port for VXLAN (default: 4789)
-vxlan-udp-sport int	UDP source port for VXLAN (default: 0)
-vxlan-vni uint	VXLAN VNI (default: 0)

Sample Commands

This section lists sample commands for some common packet types.

Table 19: Sample Packetgen Commands

Packet Type	Sample Command
ARP	packetgen -i enp0s8 -ip-ttl 32 -arp-operation 1 -progress -count 10000 -inc-smac -arp-destination-ip-address 192.168.56.1
TCP	packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac
UDP	packetgen -i enp0s8 -ip-ttl 32 -udp-sport 40000 -progress -count 10000 -inc-smac
ICMP - PING	packetgen -i enp0s8 -ip-ttl 32 -icmp-type 8 -progress -count 10000 -ip-dst 192.168.56.1
GRE	packetgen -i enp0s8 -ip-ttl 32 -gre -count 100 -inner-ip-ttl 32 -tcp-sport 3222 -progress
IP in IP	packetgen -i enp0s8 -count 100 -tcp-sport 3222 -progress -ip-src="1.1.1.1,2.2.2.2"
ETHER-IP	packetgen -i enp0s8 -ip-ttl 32 -count 100 -inner-ip-version 6 -tcp-sport 3222 -progress -inner-ethernet-smac ff:ff:ff:ff:ff:ff
VLAN	packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vlan-id 2
QinQ	packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vlan-id 2 -inner-vlan-id 2
VXLAN	packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vxlan-vni 3 -vxlan-udp-sport 4444 -inner-ip-version 4 -inner-ethernet-smac ff:ff:ff:ff:ff:ff -data-type constant
NDP	packetgen -i enp0s8 -ip-version 6 -ndp nbr-advt -count 100 -ip-checksum 1 -progress
MPLS	packetgen -i enp0s8 -ip-version 4 -mpls-label 1,2,3,4,5 -tcp-sport 4556 -count 1000 -progress

Command Example

This section shows an example command to send an ICMP ping request from source address 10.0.0.1 to destination address 10.0.0.2 via interface Hu0_0_0_25.

```
Router# bash
[ios:~]$ packetgen -i Hu0_0_0_25 -ip-ttl 32 -progress -count 50 -icmp-type 8 -ip-dst 10.0.0.2
 -ip-src 10.0.0.1 --ethernet-smac 78:c5:51:84:48:c4 --ethernet-dmac 00:00:00:1e:ca:fc
INFO[0000] [ETH IP ICMP]
INFO[0000] Setting SRC IP to 10.0.0.1
INFO[0000] Setting DST IP to 10.0.0.2
INFO[0000] Opening Handle Hu0_0_0_25
INFO[0000] Opened Handle Hu0_0_0_25
INFO[0000] Starting Packet Injection
Sending Packets... 2% | | (1/50, 254 packet/s) [0s:0s] /* Truncated output. */

Address Age Hardware Addr State Type Interface
10.0.0.1 - 78c5.5184.48c4
Interface ARPA HundredGigE0/0/0/25
10.0.0.2 00:50:23 0000.001e.cafc Dynamic ARPA HundredGigE0/0/0/25

Source stats:
Stat Name      Port Name            Control Packet Tx.  Control Packet Rx.  Ping Reply Tx.
20.0.0.2/
Card01/Port01  Ethernet - VM - 001  51                  51                  50

Interface stats:
Input    Punt XIPC  InputQ     XIPC         PuntQ
ClientID Drop/Total Drop/Total Cur/High/Max Cur/High/Max
-------------------------------------------------------------------------------
ipv6_icmp 0/0        0/0       0/0/1000     0/0/1000
icmp      0/50       0/0       0/15/1000    0/0/1000
```