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.

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 IOS XR 6.0 supports third-party off-the-shelf applications built using Linux tool chains. Users can run custom applications cross-compiled with the software development kit that Cisco provides. Cisco 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

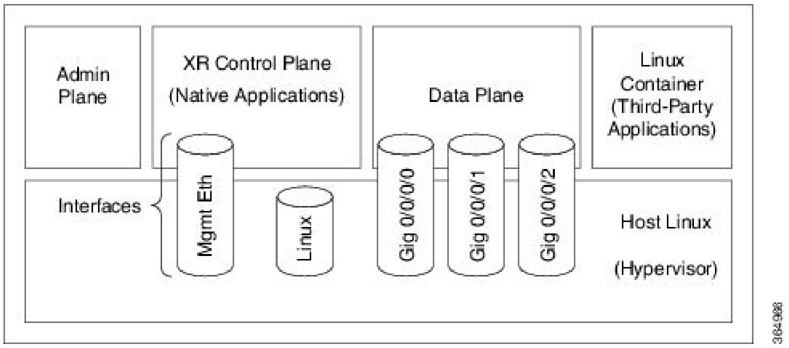
This section describes the architecture of the 64-bit IOS XR and the architecture used for application hosting.

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:

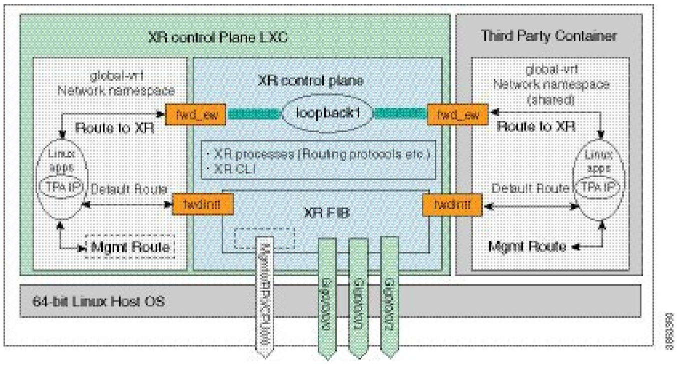
- **XR Control Plane LXC (native applications reside here)**: The XR control plane LXC contains the `global-vrf` network namespace and the XR control plane. The LXC provides the XR Linux shell to access `global-vrf` and the XR router console (CLI) to access the XR control plane. The LXC is also based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



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 4. 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 10. 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 10.

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 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.

Support for Docker Run Options via AppMgr Commands

Table 1: Feature History Table

Feature Name	Release Information	Description
Support for Docker Run Options via AppMgr Commands	Release 24.1.1	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 2: Docker Run Options

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

Docker Run Option	Description
--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	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 3: Docker Run Options

Docker Run Option	Description
--publish	Publish a container's port(s) to the host

Docker Run Option	Description
--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

The below section provides the information on how to configure the docker run time options. In this example we configure the docker run time option using appmgr and Netconf.

Step 1 Use the pids-limit command to limit the number of process IDs in the docker run command, as shown below:

Example:

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

Step 2 This step shows example for Netconf request for enabling pids-limit:

Example:

```
<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_pids_limit</source-name>
                                             <docker-run-cmd>/bin/sh</docker-run-cmd>
                                             <docker-run-opts>-it --pids-limit=10</docker-run-opts>
            </activate>
          </application>
        </applications>
      </appmgr>
    </config>
  </edit-config>
```

Verification

Procedure

	Command or Action	Purpose

Step 1	Verify the configuration - To confirm the configuration, run the following command on the router.
	Example:
	`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"`
	` !`
	`!`	

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 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.