



Application Hosting Configuration Guide for Cisco NCS 5000 Series Routers

First Published: 2015-12-23

Last Modified: 2021-07-29

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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <https://www.cisco.com/c/en/us/about/legal/trademarks.html>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2016–2021 Cisco Systems, Inc. All rights reserved.

- 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.

© 2016–2021 Cisco Systems, Inc. All rights reserved.



CONTENTS

Changes to This Document vii

CHAPTER 1

New and Changed Feature Information 1

New and Changed Application Hosting Features 1

CHAPTER 2

Getting Started with Application Hosting 3

Need for Application Hosting 3

Deep Dive Into Application Hosting 4

Application Hosting on the Cisco IOS XR Linux Shell 5

 Accessing the Third-Party Network Namespace on Cisco IOS XR Linux Shell 6

 Accessing Global VRF on the Cisco IOS XR Linux Shell 11

Getting Started with Using Vagrant for Application Hosting 15

 Accessing Global VRF on the Cisco IOS XR Linux Shell by Using a Vagrant Box 16

 Applying Bootstrap Configuration to Cisco IOS XR by Using a Vagrant Box 20

CHAPTER 3

Accessing the Networking Stack 27

Communication Outside Cisco IOS XR 27

East-West Communication for Third-Party Applications 29

Configuring Multiple VRFs for Application Hosting 31

CHAPTER 4

Hosting Applications on IOS XR 35

Types of Application Hosting 35

Native Application Hosting 36

 Setting Up the Build Environment 38

 Creating a Native Build Environment Using QEMU Hypervisor 38

 Creating a Cross-Build Environment Using the SDK Shell Script 38

Building Native RPMs	40
Running iPerf as a Native Application	42
Container Application Hosting	44
Running iPerf as a Container Application	46
Using Docker for Hosting Applications on Cisco IOS XR	48
Configuring a Docker with Multiple VRFs	49
Using Vagrant for Hosting Applications	51
Setting up an Application Development Topology By Using Vagrant	52
Deploying an Application Development Topology by Using Vagrant	54
Hosting a Wind River Linux (WRL7) Application Natively By Using Vagrant	57
Hosting an Application within a Linux Container (LXC) by Using Vagrant	64
Installing Docker on Cisco IOS XR By Using Vagrant	78

CHAPTER 5**Hosting Applications Using Configuration Management Tools 79**

Using Chef for Configuring Cisco IOS XR	79
Installing and Configuring the Chef Client	80
Creating a Chef Cookbook with Recipes	82
Using Puppet for Configuring Cisco IOS XR	83
Installing and Configuring the Puppet Agent	85
Creating a Puppet Manifest	86
Using Yang Models with Puppet on IOS XR	87
Using Configuration Management Tools on Vagrant	89
Using Puppet on Vagrant	90
Using Ansible for Hosting Applications	93
Using Ansible On Vagrant	95
Launching a Linux Container (LXC) By Using Ansible on Vagrant	98
Using Netmiko and Napalm on Vagrant	107

CHAPTER 6**Use Cases: Application Hosting 115**

Running a Telemetry Receiver in a Linux Container (LXC)	115
Use Cases on Vagrant: Container Application Hosting	119
OSPF Path Failover by Running iPerf with Netconf on Vagrant	120
Using Solenoid with exaBGP on Vagrant	127

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
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.



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, on page 1](#)

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
No new features were added.	None	Release 6.5.2	NA
No new features were added.	None	Release 6.6.2	NA
No new features were added.	None	Release 7.0.1	NA
No new features were added.	None	Release 7.1.1	NA
No new features were added.	None	Release 7.4.1	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.

- [Need for Application Hosting, on page 3](#)
- [Deep Dive Into Application Hosting, on page 4](#)
- [Application Hosting on the Cisco IOS XR Linux Shell, on page 5](#)
- [Getting Started with Using Vagrant for Application Hosting, on page 15](#)

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. Application hosting is offered in two variants: Native and Container. 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

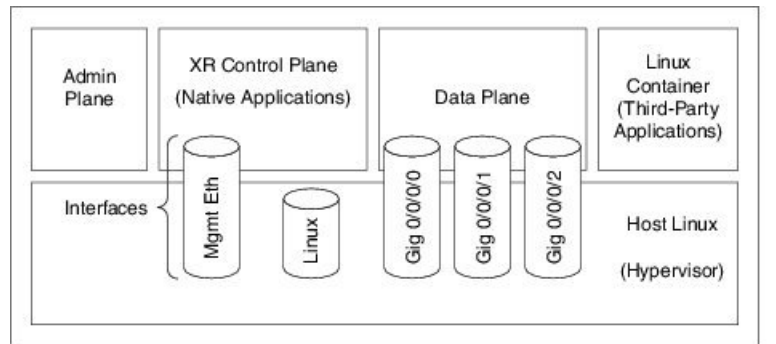
This section describes the architecture of the 64-bit IOS XR and the architecture used for application hosting.

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) is based on the Wind River Yocto distribution, and 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

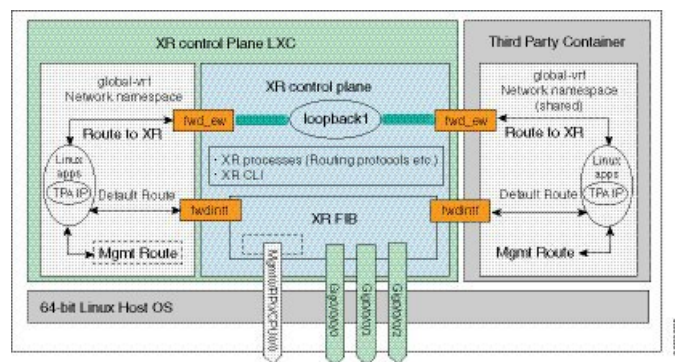
based on the WRL7 distribution. For more information on the XR control plane LXC, see [Native Application Hosting, on page 36](#).

- **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 `fw dintf` interface, which is bound to the Loopback0 interface of XR. All applications communicate with XR through the `fw d_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 6](#). Accessing the XR Linux shell takes you to the default network namespace, XRNNNS. 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 11](#). 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 XRNNNS (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 11](#).

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.

- `fw dintf` 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 `fw dintf` 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 27](#) 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 `fw dintf` and `fw dintf` interfaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fw dintf scope link src 1.1.1.1
8.8.8.8 dev fw dintf 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 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.

- `fw_d_ew` is the interface used for communication (east to west) between third-party applications and IOS XR.
- `fw_dintf` 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 `fw_dintf` 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 27](#) section.

The presence of `fw_d_ew` and `fw_dintf` 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 `fw_d_ew` and `fw_dintf` interfaces.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fw_dintf scope link src 1.1.1.1
8.8.8.8 dev fw_d_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 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>.

```
$ 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 xrapplly and xrcmd functions.
source /pkg/bin/ztp_helper.sh

function configure_xr()
{
    ## Apply a blind config
    xrapplly $1
    if [ $? -ne 0 ]; then
        echo "xrapplly 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 xrapplly 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.
- **xrapplly**: Allows you to apply (append) a configuration file to the existing configuration.
- **xrapplly_string**: Applies a configuration directly using a single inline string.
For example, **xrapplly_string "interface Gig0/0/0/0\n ip address 1.1.1.2/24 \n no shutdown**



Note To enable the **xrapplly**, **xrapplly_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-shell120160825-3292-1wncpa3.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.

- [Communication Outside Cisco IOS XR, on page 27](#)
- [East-West Communication for Third-Party Applications, on page 29](#)
- [Configuring Multiple VRFs for Application Hosting, on page 31](#)

Communication Outside Cisco IOS XR

To communicate outside Cisco IOS XR, applications use the `fw dintf` 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.

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

```

- 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 `fw dintf` 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 `fw dintf` interface to a Gigabit Ethernet interface, network connectivity can be affected if the interface goes down.

- Enter the Linux bash shell and verify whether the GigE interface address is used by the `fw dintf` 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 fw dintf scope link src 192.57.43.10
8.8.8.8 dev fw_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

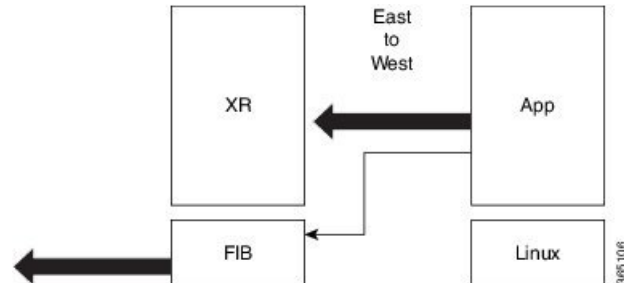
East-West communication on IOS XR is a mechanism by which applications hosted in containers interact with native XR applications (hosted in the XR control plane).

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.

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 IOS XR Release 6.3.1 supports the configuration of multiple VRFs. The applications hosted in third-party LXC's can communicate with VRFs configured on XR, after east-west communication has been enabled on the VRFs.

This section describes the configuration for creating multiple 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
  !
!

```

5. To verify if the VRFs are accessible in the IOS XR Linux shell, you can use the following commands for the respective VRF.

```

RP/0/RP0/CPU0:ios# bash vrf purple -c telnet 1.1.1.1
RP/0/RP0/CPU0:ios# bash vrf purple -c telnet 2.2.2.2
RP/0/RP0/CPU0:ios# bash vrf purple -c telnet 10::1
RP/0/RP0/CPU0:ios# bash vrf purple -c telnet 20::1
RP/0/RP0/CPU0:ios# bash vrf green -c telnet 3.3.3.3

```

```
RP/0/RP0/CPU0:ios# bash vrf green -c telnet 4.4.4.4
RP/0/RP0/CPU0:ios# bash vrf green -c telnet 30::1
RP/0/RP0/CPU0:ios# bash vrf green -c telnet 40::1
```

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, such as iPerf, can be hosted natively or in a third-party container on IOS XR.

- [Types of Application Hosting, on page 35](#)
- [Native Application Hosting, on page 36](#)
- [Container Application Hosting, on page 44](#)
- [Using Vagrant for Hosting Applications, on page 51](#)

Types of Application Hosting

Application hosting on IOS XR is offered in two variants:

- **Native:** You can host applications inside the container provided by IOS XR. Applications must be built with a Cisco-specified Linux distribution (Wind River Linux 7), which uses RPM as the package manager. The applications use the libraries found in the IOS XR root file system. Configuration management tools, such as Chef and Puppet, can be used to automate the installation of the application.
- **Container:** You can create your own container on IOS XR, 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.

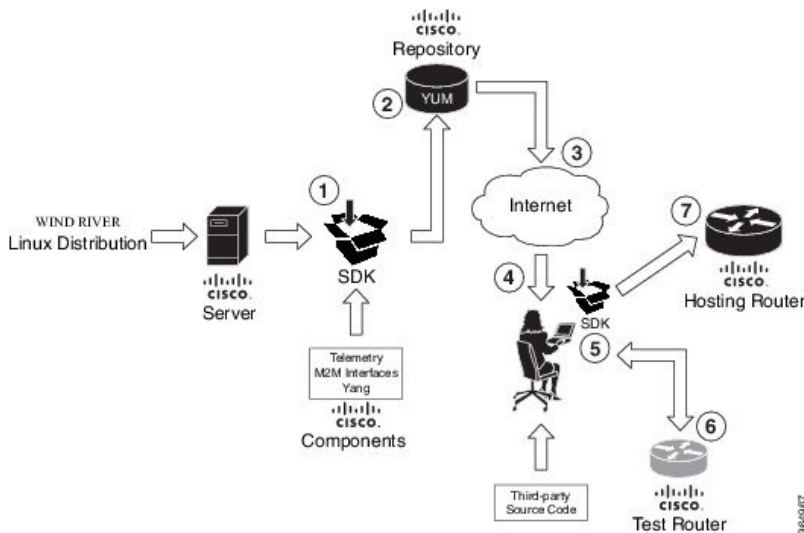
Native Application Hosting

This section describes the architecture and workflow for native application hosting.

Workflow for Native Application Hosting By Using the Shell Script

The workflow for native application hosting by using a shell script is illustrated in the following figure.

Figure 4: Native Application Hosting Using the SDK Shell Script



1. The SDK is built with Wind River Linux 7 (WRL7) distribution that uses a Yocto project workspace.

IOS XR uses the open embedded kernel infrastructure from the Yocto project as a base. Thus, IOS XR contains all the system libraries, and the basic tool chain to support applications. As a result, applications that are to be hosted natively on IOS XR must be rebuilt for the Wind River distribution and can later be customized with other IOS XR features, such as Telemetry and YANG data models.

2. The SDK is compiled and saved in the Cisco (YUM) repository.



Note The native application hosting environment is built either by using the QEMU hypervisor, or by running the SDK shell script.

Yellowdog Updated, Modified (YUM) is an open source command-line package management utility for Linux that enables creating and compiling of an embedded distribution source file into suitable formats, such as Red Hat Packet Manager (RPM). YUM allows automatic package updates. By using YUM, you can install, delete, update, and view available packages.

3. The Cisco repository is made accessible to the application developer over the Internet.
4. The developer downloads the SDK and runs the SDK shell script for installation. See [Creating a Cross-Build Environment Using the SDK Shell Script, on page 38](#) for more information.
5. The developer rebuilds the application to be hosted on IOS XR. See [Building Native RPMs, on page 40](#) for more information.

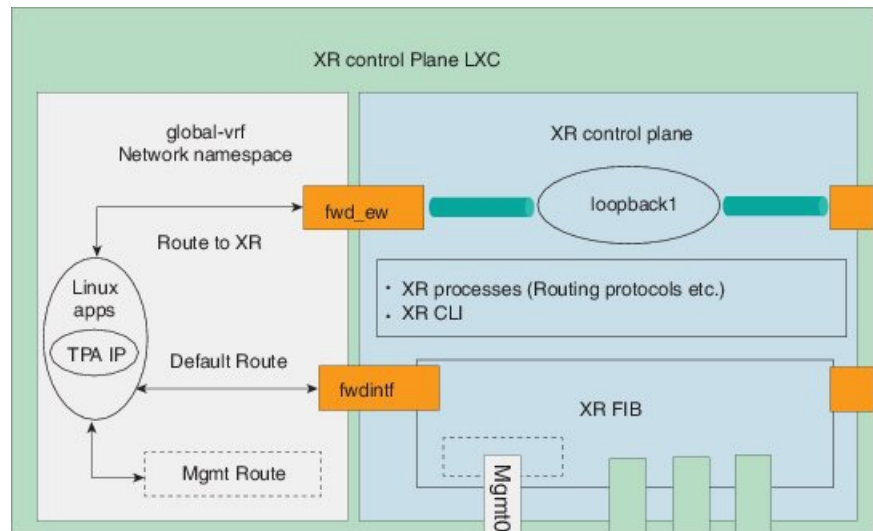
6. The developer hosts the rebuilt application on the test router.
7. The developer hosts the rebuilt application on the hosting router running IOS XR.

For information on preparing the build environment for native application hosting, see [Setting Up the Build Environment](#), on page 38.

Native Application Hosting Architecture

The architecture for native application hosting is described in the following figure.

Figure 5: Native Application Hosting Architecture



The XR control plane LXC offers the following two components:

- **global-vrf network namespace:** All native applications reside in the `global-vrf` network namespace. The applications use TPA IP address to communicate outside XR through the `fw_dintf` / Loopback0 interface of XR. You can access the namespace through the XR Linux shell. The namespace contains the following routes by default:
 - **Default route to XR FIB:** A packet with an unknown destination is sent from the Linux application to XR through the `fw_dintf/loopback0` interface.
 - **Management routes:** Routes in the subnet of the management interface, which is local to the namespace.



Note `global-vrf` is referred to as the Third-Party Network Namespace (TPNNS) in Cisco IOS XR Versions 5.3.2 and 6.0.0

- **XR control plane:** The XR control plane contains the XR routing processes and protocols and can be accessed through the XR router console (CLI). The XR FIB is programmed by the XR control plane.

Setting Up the Build Environment

This section describes the two methods of preparing and setting the build environment for native application hosting.

Creating a Native Build Environment Using QEMU Hypervisor

This section describes a method of creating the native Wind River Linux 7.0 build environment, and running the environment ISO by using a Quick Emulator (QEMU) hypervisor.

Prerequisites

- Ensure that you have access to the Cisco repository containing the native WRL7 ISO.
- Download the native ISO with the .iso extension.

Configuration Procedure

1. Launch the native WRL7 ISO and install it onto a disk image.

```
qemu-system-x86_64 -m 16G -cdrom <path-to-the-downloaded-iso-file> -net nic -net user  
-hda ./wrl7.img  
-cpu core2duo -show-cursor -usb -usbdevice wacom-tablet -vga vmware
```

2. Relaunch the native build environment with the installed image.

```
qemu-system-x86_64 -m 16G -net nic -net user -hda ./wrl7.img -cpu core2duo -show-cursor  
-usb -usbdevice wacom-tablet -vga vmware
```

The native build environment is ready for hosting third-party applications. The user is connected to the VGA console port of the native QEMU VM.

Alternatively, a user can connect to an SSH service running inside the VM.

Creating a Cross-Build Environment Using the SDK Shell Script

As an alternative to the native environment, you can create a cross-build environment by using a WRL7 cross-SDK shell script. You can install the SDK by running the shell script on a general-purpose Linux environment, such as an Ubuntu 14.04 host machine.

Prerequisites

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

- Access to the SDK in the Cisco repository.
- Ability to build any customization, as needed, on the SDK.

Installation Procedure

To install the SDK for native application hosting, use these steps:

1. Download the SDK from the Cisco repository.

```
wget https://devhub.cisco.com/artifactory/xr600/app-dev-sdk/x86_64/  
wrlinux-7.0.0.2-glibc-x86_64-intel_x86_64-wrlinux-image-glibc-std-sdk.sh
```

2. Install the SDK by running the shell script.

```
john@sjc-ads-4587:john$
./wrlinux-7.0.0.2-glibc-x86_64-intel_x86_64-wrlinux-image-glibc-std-sdk.sh
```

3. Enter the target directory for installing the SDK.

Choose a target directory that has sufficient storage space.

```
Enter target directory for SDK
(default: /opt/windriver/wrlinux/7.0-intel-x86-64):
/nobackup/john/sdk_extract
You are about to install the SDK to "/nobackup/john/sdk_extract". Proceed[Y/n]? Y
```

On successful installation, a message is displayed on the screen.

```
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

The SDK for native application hosting is successfully installed.

What to do Next

You can set up the environment variables, and validate them as explained in this section.

1. Navigate to the directory, where the SDK is installed and set up the environment variables by running the following commands:
 - If you are using a bash shell, run the `./env.sh` command.
 - For any other shell, run the `source ./env.sh` command.

The commands execute the environment setup file that was extracted during SDK installation.

2. Validate the installed environment variables by running the `env` command to view all variable values.
3. Validate the `CC` environment variable by running the `env | grep CC` command, and verifying whether the following value is assigned:

```
CC=x86_64-wrs-linux-gcc -m64
--sysroot=/nobackup/john/sdk_extract/sysroots/core2-64-wrs-linux
```

Alternatively, you can use the `echo` command:

```
echo $CC
x86_64-wrs-linux-gcc -m64
--sysroot=/opt/windriver/wrlinux/7.0-intel-x86-64/sysroots/core2-64-wrs-linux
```

4. Verify whether the `PATH` environment variable points to the base directory, where the SDK was installed. To verify the path, run the `env | grep PATH` command and check whether the following path is displayed:

```
PATH=<sdk_extract>/sysroots/
x86_64-wrlinuxsdk-linux/usr/bin:
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin/x86_64-wrs-linux
```

Alternatively, you can use the `echo` command:

```
echo $PATH
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin:
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin/x86_64-wrs-linux
```

5. Navigate to the directory that contains the application source code, and start building the application.



Note You should remove all the *.la temporary files from the SDK root file system. To do this, use the following commands:

```
bash# cd <sdk_extract>/sysroots/  
bash# find . -name \*.la | xargs rm -f
```

Building Native RPMs

This section describes the procedure for building applications by using either the native environment, or the cross-build environment. It is recommended that you use the native build environment.

There are two ways of building applications from source code.

One method is to build an application from a source code archive; this is explained in this section. The other method is to build it from a source RPM, which is not recommended.

Prerequisites

Ensure that the following requirements are met before you proceed:

- The application build environment has been set up to use either the native build environment, or the cross-build environment.
- You have read the README file to understand the build process for building the application.

Configuration Procedure

To build applications, use the following steps:

1. Navigate to the directory that contains the source code for the application.
2. Run the following commands to extract the application (if compressed).

```
bash-4.1$ tar xzvf tcpdump-4.7.4.tar.gz
```

3. Change your directory to the application directory.

```
bash-4.1$ cd tcpdump-4.7.4
```

4. Build your application to generate an executable file.

```
tcpdump-4.7.4$ ./configure  
tcpdump-4.7.4$ make
```

5. Verify the executable file in your directory.

```
tcpdump-4.7.4$ ls -l ./tcpdump  
-rwxr-xr-x 1 john eng 3677288 Jun 15 23:10 ./tcpdump
```

The executable file is listed as `tcpdump`.

The executable file is ready to be packaged for hosting your application on IOS XR.

What to do Next

Package the application binaries so that it can be installed on IOS XR.

The recommended packaging format is RPM so that it can be hosted on IOS XR.

To build an RPM:

- You need a .spec file.
- You must run the **rpmbuild** command.

Use the following steps to package the binaries:

1. Create a .spec file in the SPECS directory.

```
# %define __strip /bin/true

Name: tcpdump
Version: 4.7.4
Release: XR
Buildroot: %{_tmppath}/%{name}-%{version}-%{release}-root
License: Copyright (c) 2015 Cisco Systems Inc. All rights reserved.
Packager: mark
SOURCE0 : %{name}-%{version}.tar.gz
Group: 3'rd party applicaiton
Summary: Tcpdump cross compiled for WRL6

%description
This is a cross compiled version of tcpdump using IOS XR sdk for WRL7

%prep

%setup -q -n %{name}-%{version}

%build
# This where sdk is being sourced
source /nobackup/mark/sdk_extract_18/tmp/env.sh
./configure
make

%install
rm -rf ${RPM_BUILD_ROOT}
# make DESTDIR=${RPM_BUILD_ROOT} install
mkdir -p ${RPM_BUILD_ROOT}%{_sbindir}
install -m755 tcpdump ${RPM_BUILD_ROOT}%{_sbindir}

%files
%defattr(-,root,root)
%{_sbindir}/tcpdump

%pre

%post

%preun

%postun

%clean
rm -rf $RPM_BUILD_ROOT
```

2. Build the RPM.

```
mark@tenby:redhat$ cd /usr/src/redhat/SPECS/
mark@tenby:SPECS$ rpmbuild -ba tcpdump.spec
```

The RPM build used is the 5.4.14 version.

- Verify that the binary is built in the RPMS directory.

```
mark@tenby:x86_64$ pwd /usr/src/redhat/RPMS/x86_64
mark@tenby:x86_64$ ls
tcpdump-4.7.4-XR.x86_64.rpm
```

The native applications are ready to be hosted. For information on hosting native applications, see [Running iPerf as a Native Application, on page 42](#).

Running iPerf as a Native Application

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

Topology

This figure illustrates the topology used in this example.

Figure 6: iPerf as a Native Application



iPerf server is installed on Router A, and iPerf client is installed on Router B. Both installations are done natively on 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 native application, follow these steps:

- Log into Router A, and enter the XRNS.

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

- Install the iPerf server as an RPM on Router A.

```
[xr-vm_node0_RP0_CPU0:~]$ yum install
https://devhub.cisco.com/artifactory/xr600/3rdparty/x86_64/iperf-2.0.5-r0.0.core2_64.rpm
```

- Perform Steps 1 and 2 to install the iPerf client on Router B.

- Verify the iPerf server installation on Router A.

```
[xr-vm_node0_RP0_CPU0:~]$ iperf -v

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

Similarly, verify the iPerf client installation on Router B.

5. 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.

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

6. 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.

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

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

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

```
[xr-vm_node0_RP0_CPU0:~]$ /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 accesses the iPerf server hosted on Router A.

Native application hosting for iPerf is successful.

iPerf Operations on IOS XR By Using System V Init Script

The iPerf server or client service can be started, stopped, or restarted automatically on IOS XR. To achieve this, a System V (SysV) script must be created, and added as shown in this example:

1. Create the SysV script for starting, stopping, or reloading the iPerf server or client.

In this example, the script is created for the iPerf server. To create the script for the iPerf client, instead, use the command described in Step 5 of the iPerf configuration procedure.

```

#!/bin/bash
#
# description: iPerf server
#
# Get function from functions library
. /etc/init.d/functions

# Start the service iperf
start() {
    iperf -s -B 1.1.1.1 -p 57730 & "Starting the iPerf Server: "
    /path/to/iperf &
    ### Create the lock file ###
    touch /var/lock/subsys/iperf
    success $"iPerf server startup"
    echo
}

# Restart the service iperf
stop() {
    iperf -s -B 1.1.1.1 -p 57730 & "Stopping the iPerf Server: "
    killproc iperf
    ### Delete the lock file ###
    rm -f /var/lock/subsys/iperf
    echo
}

### main logic ###
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status iperf
        ;;
    restart|reload|condrestart)
        stop
        start
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac
exit 0

```

2. Add the script to IOS XR hosting the iPerf server or client.

```
bash-4.3# chkconfig --add iperf
```

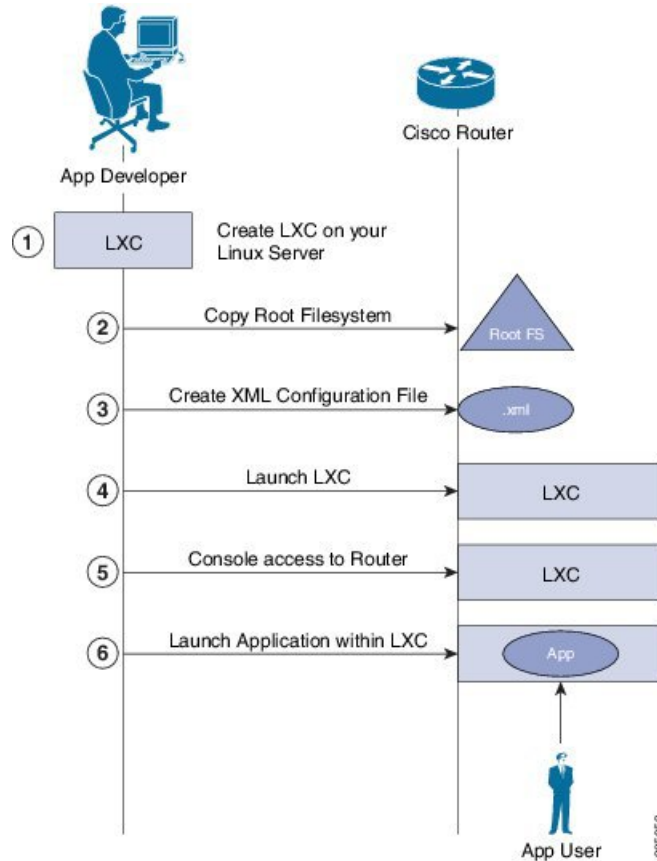
Container Application Hosting

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 46](#).

Figure 7: 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 8: 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.

Container application hosting for iPerf is successful. For information on a use case of container application hosting, see [Running a Telemetry Receiver in a Linux Container \(LXC\)](#), on page 115.

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.

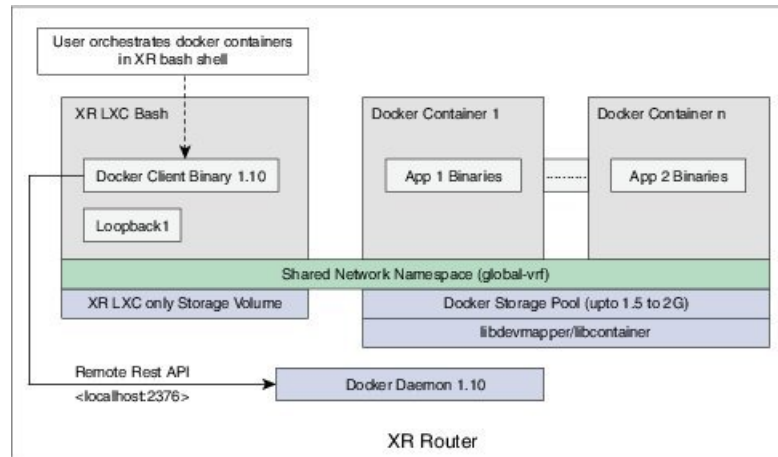
Docker follows a layered approach that consists of a base image at the bottom that supports layers of applications on top. The base images are available publicly in a repository, depending on the type of application you want to install on top. You can manipulate docker images by using the docker index 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 9: Docker on IOS XR

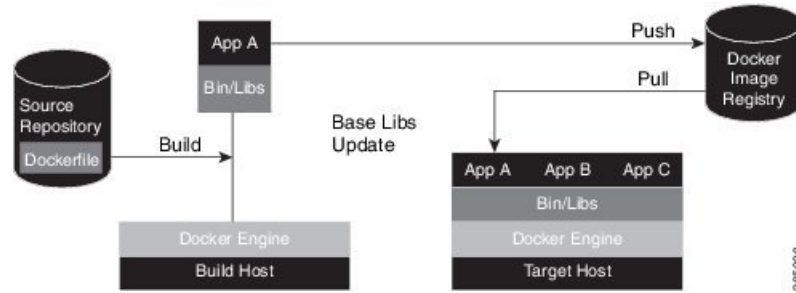


The docker containers are created alongside the LXC on XR. To create and manage the containers, you can use the XR bash shell. This is where the docker client is installed. The application binaries for the applications to be hosted are installed inside the docker container.

Hosting Applications in Docker Containers

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

Figure 10: Docker Workflow for Application Hosting

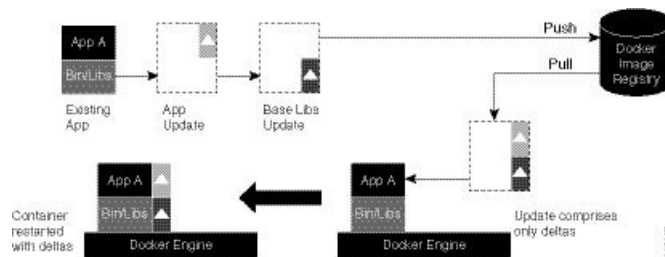


1. The docker file in the source repository is used to build the application binary file on your (docker engine build) host machine.
2. The application binary file is pushed into the docker image registry.
3. The application binary file is pulled from the docker image registry and copied to the docker 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 11: 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.

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 31.

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 privileges 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.

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.

Pre-requisites for Using Vagrant

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

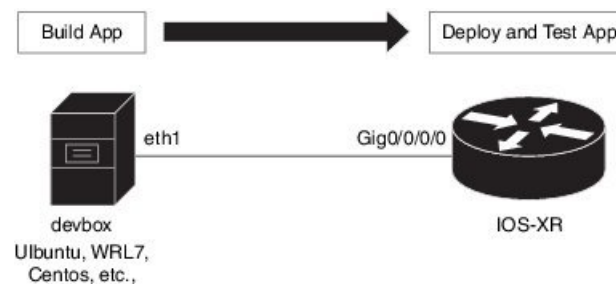
- Latest version of [Vagrant](#) for your operating system. We recommend Version 1.8.6.
- Latest version of a [virtual box](#) for your operating system. We recommend Version 5.1+.
- Minimum of 5 GB of RAM with two cores.

- (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 12: 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
```

```

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 54](#) 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 52](#), 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.

- 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).

- 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.

- 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:~]$
    
```

- 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.

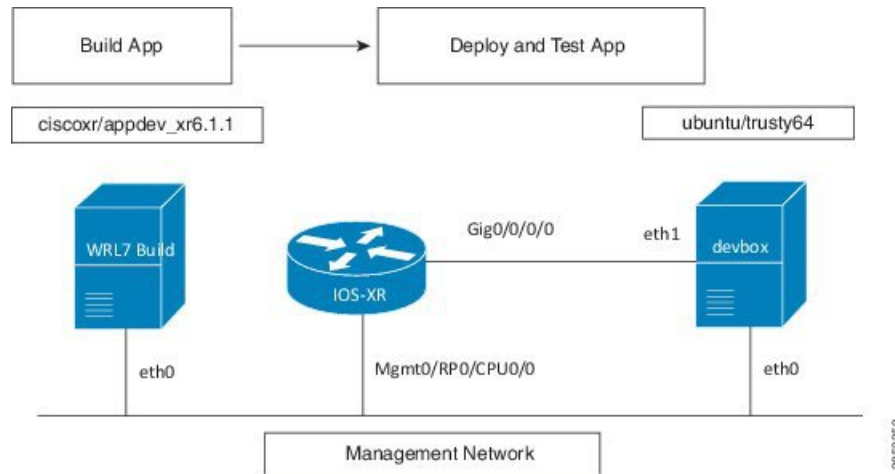
Hosting a Wind River Linux (WRL7) Application Natively By Using Vagrant

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 13: 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 52](#), 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.
2. Clone the `vagrant-xrdocs` repository.
3. Navigate to the `vagrant-xrdocs/native-app-topo-bootstrap` directory and launch the vagrant instance.

```
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)
```

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

```
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!
...
==> wr17_build: Checking if box 'ciscoxr/appdev-xr6.1.1' is up to date...
==> wr17_build: Clearing any previously set forwarded ports...
==> wr17_build: Fixed port collision for 22 => 2222. Now on port 2201.
==> wr17_build: Clearing any previously set network interfaces...
==> wr17_build: Preparing network interfaces based on configuration...
      wr17_build: Adapter 1: nat
==> wr17_build: Forwarding ports...
      wr17_build: 22 (guest) => 2201 (host) (adapter 1)
==> wr17_build: Booting VM...
==> wr17_build: Waiting for machine to boot. This may take a few minutes...
      wr17_build: SSH address: 127.0.0.1:2201
      wr17_build: SSH username: vagrant
      wr17_build: SSH auth method: private key
      wr17_build: Warning: Remote connection disconnect. Retrying...
...
==> wr17_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) rtdld(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 wr17_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.

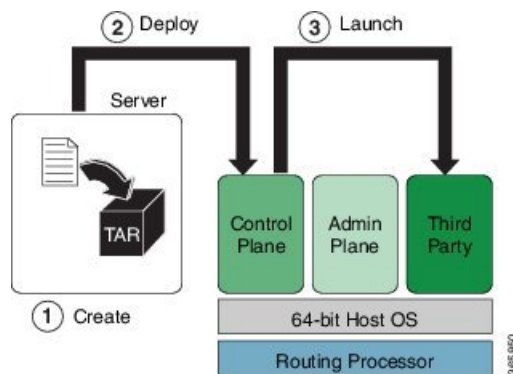
Hosting an Application within a Linux Container (LXC) by Using Vagrant

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 14: 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 52](#), 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/ubuntu.com trusty InRelease
Get:1 http://security.ubuntu.com/ubuntu.com trusty-security InRelease [65.9 kB]
...
Get:33 http://archive.ubuntu.com/ubuntu.com trusty-backports/universe Translation-en [36.8 kB]
Hit http://archive.ubuntu.com/ubuntu.com trusty Release
...
Hit http://archive.ubuntu.com/ubuntu.com trusty/universe Translation-en
Ign http://archive.ubuntu.com/ubuntu.com trusty/main Translation-en_US
Ign http://archive.ubuntu.com/ubuntu.com trusty/multiverse Translation-en_US
Ign http://archive.ubuntu.com/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
  libmn10 libnetfilter-contrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbdl libseccomp2 libxslt1.1 lxc 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 lxcctl 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
  libmn10 libnetfilter-contrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbdl 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 librbdl (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>
<sharenets 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 `sharenets` 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:ad:eb/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)

fdintf  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 -zxvf
../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 **J** 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.



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 79](#)
- [Using Puppet for Configuring Cisco IOS XR, on page 83](#)
- [Using Configuration Management Tools on Vagrant, on page 89](#)

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 3: 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

The following subsections describe how you can install, configure, and create Chef recipes for native application hosting. For information on native application hosting, see [Native Application Hosting, on page 36](#).

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/wr17_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.

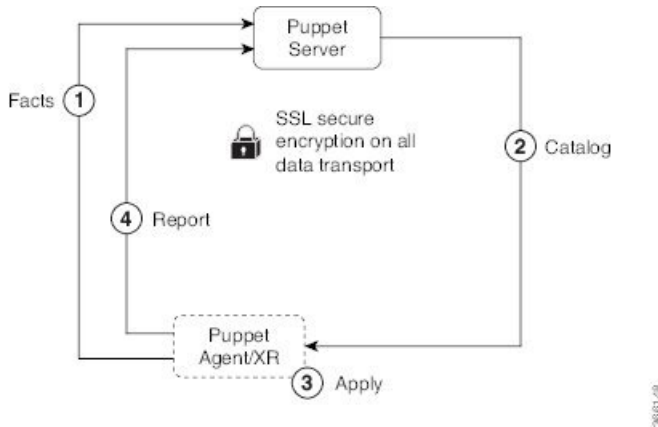
For additional details on the Chef Client, refer to https://docs.chef.io/chef_client.html

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 language 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 15: 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 4: 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

The subsections that follow describe how you can install, configure, and create Puppet manifests for native application hosting. For information on native application hosting, see [Native Application Hosting, on page 36](#).

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)

fdintf 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
}
```

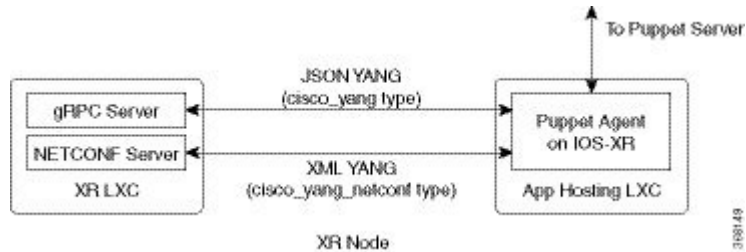
2. Access and trigger the Puppet Agent to converge the system based on the manifest defined on the Puppet Server.

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 16: 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:vrf": [null]}',
    source => '{"Cisco-IOS-XR-infra-rsi-cfg:vrf": {
      "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 => '<vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg"/>',
    source => '<vrf 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>
    </vrf/>',
    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 5: 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	<ul style="list-style-type: none"> • /pkg/bin/xr_cli • /pkg/sbin/config

Mode of Access	Method of Access	SSH Port Number	Helper Programs
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	<ul style="list-style-type: none"> • /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 52](#), 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",
    "      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.

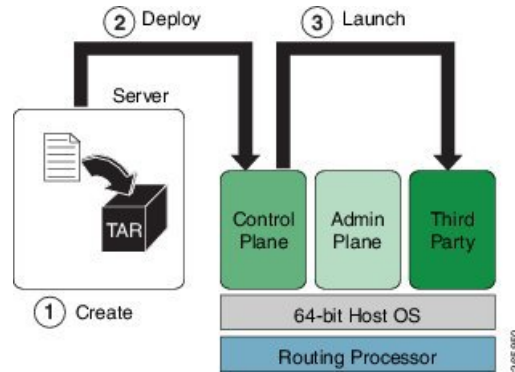
Launching a Linux Container (LXC) By Using Ansible on Vagrant

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 17: 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 52](#), 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 libaiol libboost-system1.54.0 libboost-thread1.54.0 liblxc1
  libmn10 libnetfilter-contrack3 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 lxcctl 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 libaiol libboost-system1.54.0 libboost-thread1.54.0 liblxc1
  libmn10 libnetfilter-contrack3 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 libaiol 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 libaiol:amd64.
(Reading database ... 62989 files and directories currently installed.)
Preparing to unpack .../libaiol_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.

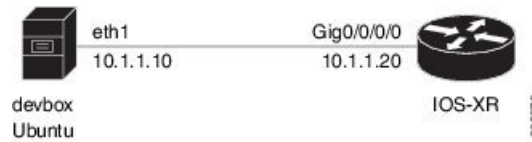
Using Netmiko and Napalm on Vagrant

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 18: 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

Use Cases: Application Hosting

This chapter describes use cases for running applications on IOS XR.

- [Running a Telemetry Receiver in a Linux Container \(LXC\)](#), on page 115

Running a Telemetry Receiver in a Linux Container (LXC)

For telemetry to work on Cisco IOS XR, it must use GPB (Google Protocol Buffer) over UDP, instead of TCP.

The procedure consists of the following steps:

1. Create a telemetry policy file.
2. Generate and compile a .proto file.
3. Configure the GPB encoder.
4. Launch a third-party container (LXC).
5. Configure the telemetry receiver.

Creating a Telemetry Policy File

A telemetry policy file is used to specify the kind of data to be generated and pushed to the telemetry receiver. The following steps describe how you can create the policy file for telemetry:

1. Determine the schema paths to stream data.

```
RP/0/RP0/CPU0:ios# schema-describe show interface  
Wed Aug 26 02:24:40.556 PDT  
RootOper.InfraStatistics.Interface(*).Latest.GenericCounters
```

2. Create a policy file that contains these paths:

```
{  
  "Name": "Test",  
  "Metadata": {  
    "Version": 25,  
    "Description": "This is a sample policy",  
    "Comment": "This is the first draft",  
    "Identifier": "<data that may be sent by the encoder to the mgmt stn"  
  },  
  "CollectionGroups": {
```

```
"FirstGroup": {
  "Period": 30,
  "Paths": [
    "RootOper.InfraStatistics.Interface(*) .Latest.GenericCounters"
  ]
}
}
```

3. Enter the XR Linux bash shell, and copy the policy file to IOS XR by using Secure Copy Protocol (SCP).

```
/* 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:~]$ scp Test.policy cisco@10.0.0.1:/telemetry/policies
cisco@10.0.0.1's password:
Test.policy
100% 779 0.8KB/s 00:00
Connection to 10.0.0.1 closed by remote host.
```

Where 10.0.0.1 is the IP address of the device on which you are copying the policy file.

4. Navigate to the IOS XR prompt and verify if the policy file has been successfully installed.

```
RP/0/RP0/CPU0:ios# show telemetry policies brief
Wed Aug 26 02:24:40.556 PDT
Name |Active?| Version | Description
-----|-----|-----|-----
Test N 1 This is a sample policy
```

Generating and Compiling a .proto File

The path in a policy file that you created needs a `.proto` file associated with it. The `.proto` file describes the GPB message format used to stream data. The following steps describe how you can generate and compile a `.proto` file for a telemetry receiver:

The `.proto` file is compiled into a `.map` file. The compilation is done on a server.

1. Generate a `.proto` file.

```
telemetry generate gpb-encoding path
"RootOper.InfraStatistics.Interface(*) .Latest.GenericCounters" file
disk0:generic_counters.proto
```

The `.proto` file is generated by an on-box tool. The tool ignores naming parameters, and are hence optional.



Note The tool ignores text within quotes; therefore, the path should not contain quotes.

2. Compile the `.proto` file off the box.

- a. Cisco provides a telemetry compiler on Dev Hub. You can copy the directory to your Linux box, and run it, as shown here:

```
telemetry_protoc -f generic_counters.proto -o generic_counters.map
```

- b. Access the copy of the .proto file from Dev Hub, and run the standard compiler on your Linux box, as shown here:

```
protoc python_out . -I=/
sw/packages/protoc/current/google/include/:.
generic_counters.proto ipv4_counters.proto
```

3. Copy the map file to IOS XR at /telemetry/gpb/maps.

Configuring the GPB Encoder

Configure the GPB encoder to activate the telemetry policy and stream data as outlined in the following steps:

1. Configure a loopback interface address for mapping the telemetry receiver to IOS XR, as shown here:

```
RP/0/RP0/CPU0:ios(config)# interface Loopback2
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 2.2.2.2/32
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
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
Loopback2	2.2.2.2	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

2. Configure the encoder to stream the policy to the loopback interface of IOS XR that was just configured.

```
telemetry
  encoder gpb
  policy group alpha
  policy demo
  destination ipv4 2.2.2.2 port 5555
  !
!
```

Launching a Third-Party Container (LXC)

This section describes how you can launch a third-party container (LXC) on IOS XR.

1. Log into IOS XR.

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

2. Launch the third-party container.

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

3. Log into the container 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:
```

You have successfully launched a third-party container.

Configuring the Telemetry Receiver

A telemetry receiver listens for streamed data on the specified interface IP address and port number, and it prints the header of the received packets. If .proto files are provided, they are compiled using the protoc compiler and the message contents are also printed. By default, only the first row of each table is printed, though the `print-all` option can be used to print the complete output.

To run a telemetry receiver within the container you launched, use the following steps:

1. Download all the receiver files to the third-party container. The receiver files are available on IOS XR at <https://github.com/cisco/bigmuddy-network-telemetry-collector>.
2. Run the receiver to stream and print data.

```
python gpb_receiver.py ipaddress 2.2.2.2 port 5555 proto
generic_counters.proto ipv4_counters.proto
```

You can see data on the telemetry receiver, as shown here:

```
Waiting for message
Got message of length:1036bytes from address:('10.1.1.1', 5555)
Encoding:2271560481
Policy Name:demo
Version:25
Identifier:<data that may be sent by the encoder to the mgmt stn>
Start Time:Wed Jan 21 09:54:33 1970
End Time:Wed Aug 26 09:28:37 2015
# Tables:1
Schema
Path:RootOper.InfraStatistics.Interface.Latest.GenericCounters
# Rows:6
Row 0:
applique:0
availability_flag:0
broadcast_packets_received:0
broadcast_packets_sent:0
bytes_received:0
bytes_sent:0
carrier_transitions:0
crc_errors:0
framing_errors_received:0
giant_packets_received:0
input_aborts:0
input_drops:0
input_errors:0
input_ignored_packets:0
input_overruns:0
input_queue_drops:0
interface_name:Null0
last_data_time:1440606516
last_discontinuity_time:1440498130
multicast_packets_received:0
multicast_packets_sent:0
output_buffer_failures:0
output_buffers_swapped_out:0
output_drops:0
output_errors:0
output_queue_drops:0
```

```
output_underruns:0
packets_received:0
packets_sent:0
parity_packets_received:0
resets:0
runt_packets_received:0
seconds_since_last_clear_counters:0
seconds_since_packet_received:4294967295
seconds_since_packet_sent:4294967295
throttled_packets_received:0
unknown_protocol_packets_received:0
Waiting for message
Got message of length:510bytes from address:('2.2.2.2', 5555)
Encoding:2271560481
Policy Name:demo
Version:25
Identifier:<data that may be sent by the encoder to the mgmt stn>
Start Time:Wed Jan 21 09:54:33 1970
End Time:Wed Aug 26 09:28:38 2015
# Tables:1
Schema Path:RootOper.InfraStatistics.Interface.Latest.Protocol
# Rows:5
Row 0:
bytes_received:0
bytes_sent:0
input_data_rate:0
input_packet_rate:0
interface_name:Loopback2
last_data_time:1440606517
output_data_rate:0
output_packet_rate:0
packets_received:0
packets_sent:0
protocol:24
protocol_name:IPV4_UNICAST
```

The telemetry receiver runs successfully within the third-party container (LXC).

Use Cases on Vagrant: Container Application Hosting

This section describes how you can use vagrant to run use cases for container application hosting.

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.

OSPF Path Failover by Running iPerf with Netconf on Vagrant

This section describes a use case for solving a path remediation problem by using iPerf and Netconf applications on vagrant.

Topology

The topology used for OSPF path remediation is illustrated in the following figure.

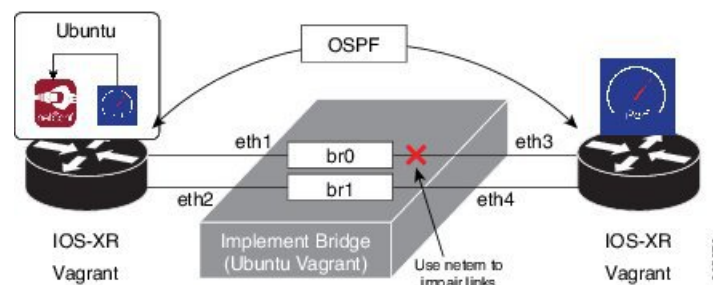
The router on the left is rtr1 and is the source of traffic. We run the pathchecker application inside an LXC on this router. Pathchecker uses an iPerf client to determine the health of the path.

The router on the right is rtr2 and is the destination for traffic. We run the pathchecker application inside an LXC on this router. Pathchecker uses an iPerf server that talks to the iPerf client on rtr1.

devbox serves two purposes in this topology:

- To create an LXC tar ball with pathchecker before being deployed to the routers.
- To bridge the two networks between the two routers over the parallel paths.

Figure 19: OSPF Path Failover with iPerf and Netconf on Vagrant



This example uses the following process for OSPF path failover:

1. Configure and establish OSPF neighbor relationship between two routers over two parallel paths.
2. Increase the cost of one path so that the other path is the preferred active path.
3. Use the pathchecker python application to monitor the OSPF active path by determining the bandwidth, jitter, packet loss and other parameters. Pathchecker uses the iPerf application to measure health of the active traffic path.
4. Use pathchecker to simulate network degradation by changing the OSPF active path cost during a Netconf session.

Procedure

Use the following steps to use iPerf with Netconf for OSPF path failover.

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.

```
AKSHSHAR-M-KODS:~ akshshar$ vagrant box list
IOS-XRv (virtualbox, 0)
```

4. Create a working directory.

```
AKSHSHAR-M-KODS:~ akshshar$ mkdir ~/iosxrv
AKSHSHAR-M-KODS:~ akshshar$ cd ~/iosxrv
```

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

```
AKSHSHAR-M-KODS:~ akshshar$ 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 repository containing the pathchecker code.

```
AKSHSHAR-M-KODS:~ akshshar$ git clone https://github.com/ios-xr/pathchecker.git
Cloning into 'pathchecker'...
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 46 (delta 8), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (46/46), done.
Checking connectivity... done.
```

7. Navigate to the pathchecker/vagrant directory and launch devbox.

```
AKSHSHAR-M-KODS:~ akshshar$ cd pathchecker/
AKSHSHAR-M-KODS:pathchecker akshshar$ cd vagrant/
AKSHSHAR-M-KODS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant

AKSHSHAR-M-KODS:vagrant akshshar$ vagrant up devbox
Bringing machine 'devbox' up with 'virtualbox' provider...
==> devbox: Importing base box 'ubuntu/trusty64'...
```

```
----- snip output -----
```

```
==> devbox: Running provisioner: file...
AKSHSHAR-M-KODS:vagrant akshshar$
AKSHSHAR-M-KODS:vagrant akshshar$
AKSHSHAR-M-KODS:vagrant akshshar$ vagrant status
Current machine states:
```

```
rtr1                not created (virtualbox)
devbox              running (virtualbox)
rtr2                not created (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. Launch an LXC within devbox.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh devbox

vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-create -t ubuntu --name pathchecker
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
I: Checking Release signature
...
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-start --name pathchecker
<4>init: hostname main process (3) terminated with status 1
<4>init: plymouth-upstart-bridge main process (5) terminated with status 1
<4>init: plymouth-upstart-bridge main process ended, respawning
```

Ubuntu 14.04.4 LTS nc_iperf console

pathchecker login: ubuntu

Password:

Welcome to Ubuntu 14.04.4 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.

...

9. Install all the required iPerf and Netconf application dependencies within the LXC.

```
ubuntu@pathchecker:~$ sudo apt-get -y install python-pip python-lxml
python-dev libffi-dev libssl-dev iperf git

ubuntu@pathchecker:~$ sudo pip install ncclient jinja2 cryptography==1.2.1
```

10. Retrieve the iPerf and Netconf application code from Github.

```
ubuntu@pathchecker:~$ git clone https://github.com/ios-xr/pathchecker.git
Cloning into 'pathchecker'...
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 46 (delta 8), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (46/46), done.
Checking connectivity... done.
ubuntu@pathchecker:~$
```

11. Change the SSH port inside the LXC.

When a container is deployed on XR, it shares the network namespace of XR. Since XR uses ports 22 and 57722 for internal processes, we change the port number to 58822 in this example.

```
ubuntu@pathchecker:~$ sudo sed -i s/Port\ 22/Port\ 58822/ /etc/ssh/sshd_config

ubuntu@pathchecker:~$ cat /etc/ssh/sshd_config | grep Port
Port 58822
```

12. Create the LXC tar ball.
 - a. Shut down the LXC.

```
ubuntu@pathchecker:~$ sudo shutdown -h now
ubuntu@pathchecker:~$
Broadcast message from ubuntu@pathchecker
(/dev/lxc/console) at 10:24 ...
```

The system is going down for halt NOW!

- b. Assume the root user role.

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

- c. Navigate to the `/var/lib/lxc/pathchecker/rootfs/` directory and package the `rootfs` into a tar ball.

```
root@vagrant-ubuntu-trusty-64:~# cd /var/lib/lxc/pathchecker/rootfs/
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/pathchecker/rootfs/# tar -czvf
/vagrant/pathchecker_rootfs.tar.gz *
tar: dev/log: socket ignored
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/pathchecker/rootfs/# exit
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.
```

```
AKSHSHAR-M-K0DS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant
AKSHSHAR-M-K0DS:vagrant akshshar$ ls -l pathchecker_rootfs.tar.gz
-rw-r--r-- 1 akshshar staff 301262995 Jul 18 07:57 pathchecker_rootfs.tar.gz
AKSHSHAR-M-K0DS:vagrant akshshar$
```

13. Launch the two router topology.

- a. Navigate to the `pathchecker/vagrant` directory and launch the vagrant instance.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant

AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant up
Bringing machine 'rtr1' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr2' up with 'virtualbox' provider...
```

- b. Verify if the topology has been launched.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant status
Current machine states:
```

```
rtr1                running (virtualbox)
devbox              running (virtualbox)
rtr2                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``.

14. Verify if OSPF is running on `rtr1` and check the path state.

You can also see the cost of the OSPF path.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant port rtr1
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) => 2200 (host)
58822 (guest) => 58822 (host)
AKSHSHAR-M-K0DS:vagrant akshshar$ ssh -p 2223 vagrant@localhost
The authenticity of host '[localhost]:2223 ([127.0.0.1]:2223)' can't be established.
RSA key fingerprint is bl:c1:5e:a5:7e:e7:c0:4f:32:ef:85:f9:3d:27:36:0f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2223' (RSA) to the list of known hosts.
vagrant@localhost's password:
```

```
RP/0/RP0/CPU0:rtr1# show running-config router ospf
Mon Jul 18 15:25:53.875 UTC
router ospf apphost
 area 0
  interface Loopback0
  !
  interface GigabitEthernet0/0/0/0
  !
  interface GigabitEthernet0/0/0/1
  cost 20
  !
  !
  !

RP/0/RP0/CPU0:rtr1# show route 2.2.2.2
Mon Jul 18 15:26:03.576 UTC

Routing entry for 2.2.2.2/32
  Known via "ospf apphost", distance 110, metric 2, type intra area
  Installed Jul 18 15:18:28.218 for 00:07:35
  Routing Descriptor Blocks
    10.1.1.20, from 2.2.2.2, via GigabitEthernet0/0/0/0
      Route metric is 2
  No advertising protos.
RP/0/RP0/CPU0:rtr1#
```

15. Start the iPerf server on `rtr2` and configure it for receiving packets from `rtr1`.



Note iPerf was launched as a native application on `rtr2` while launching the vagrant instance.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh rtr2
Last login: Mon Jul 18 15:57:05 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$
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)
```

16. Launch the pathchecker application within the LXC on `rtr1`.
 - a. Log in to the LXC on `rtr1`.

Password for user `ubuntu` is **ubuntu**.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ ssh -p 58822 ubuntu@localhost
The authenticity of host '[localhost]:58822 ([127.0.0.1]:58822)' can't be
established.
RSA key fingerprint is 19:54:83:a9:7a:9f:0a:18:62:d1:f3:91:87:3c:e9:0b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:58822' (RSA) to the list of known hosts.
ubuntu@localhost's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.14.23-WR7.0.0.2_standard x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Mon Jul 18 15:19:45 2016 from 10.0.2.2
ubuntu@pathchecker:~$
```

- b. Navigate to the pathchecker repository within the LXC, and check the contents of the pathchecker script.

```
ubuntu@pathchecker:~$ cd pathchecker/
ubuntu@pathchecker:~/pathchecker$ cat pc_run.sh
#!/bin/bash

./pathchecker.py --host 6.6.6.6 -u vagrant -p vagrant --port 830 -c 10 -o
apphost -a 0 -i GigabitEthernet0/0/0/0 -s 2.2.2.2 -j 4 -l 5 -f -t 10
```

`-l` represents the threshold for packet loss and has been set to 5% for this run. `-j` represents the jitter threshold that has a value of 4.

- c. Start the pathchecker application by running the script.

```
ubuntu@pathchecker:~/pathchecker$ ./pc_run.sh
Error while opening state file, let's assume low cost state
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run.....
20160718162513,1.1.1.1,62786,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718162513,1.1.1.1,62786,2.2.2.2,5001,6,0.0-10.0,1312710,1048474
20160718162513,2.2.2.2,5001,1.1.1.1,62786,6,0.0-10.0,1312710,1048679,2.453,0,892,0.000,1

bw is
1025.5546875
jitter is
2.453
pkt_loss is
0.000
verdict is
False
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run.....
```

The pathchecker application is running on the path from `GigabitEthernet0/0/0/0` interface.

17. Open a parallel Git bash window and simulate impairment on the active path.

- a. Access devbox through SSH.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ cd pathchecker/vagrant
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

...

```

- b. View the impairment script and run it on `devbox`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ls
impair_backup.sh  impair_reference.sh  stop_impair.sh

vagrant@vagrant-ubuntu-trusty-64:~$ cat impair_reference.sh
#!/bin/bash
echo "Stopping all current impairments"
sudo tc qdisc del dev eth3 root &> /dev/null
sudo tc qdisc del dev eth4 root &> /dev/null
echo "Starting packet loss on reference link"
sudo tc qdisc add dev eth3 root netem loss 7%

vagrant@vagrant-ubuntu-trusty-64:~$ ./impair_reference.sh
Stopping all current impairments
Starting packet loss on reference link
```

The script creates a packet loss of 7% on the reference link.

18. Open the first Git bash window to view the pathchecker application running on `rtr1`.

```
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run....
20160718164745,1.1.1.1,60318,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164745,1.1.1.1,60318,2.2.2.2,5001,6,0.0-10.0,1312710,1048516
20160718164745,2.2.2.2,5001,1.1.1.1,60318,6,0.0-573.0,1312710,18328,5.215,0,892,0.000,1

bw is
1025.5546875
jitter is
5.215
pkt_loss is
0.000
verdict is
True
Woah! iperf run reported discrepancy, increase cost of reference link !
Increasing cost of the reference link GigabitEthernet0/0/0/0
Currently, on backup link
Starting an iperf run....
20160718164755,1.1.1.1,61649,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164755,1.1.1.1,61649,2.2.2.2,5001,6,0.0-10.0,1312710,1048577
20160718164755,2.2.2.2,5001,1.1.1.1,61649,6,0.0-583.3,1312710,18002,1.627,0,893,0.000,0

bw is
1025.5546875
jitter is
1.627
pkt_loss is
0.000
verdict is
False
Currently, on backup link
Starting an iperf run....
20160718164805,1.1.1.1,59343,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164805,1.1.1.1,59343,2.2.2.2,5001,6,0.0-10.0,1312710,1048520
20160718164805,2.2.2.2,5001,1.1.1.1,59343,6,0.0-593.4,1312710,17697,2.038,0,893,0.000,0
```

Pathchecker has initiated a failover from primary to secondary link.

19. Verify if the failover was successful on `rtr1`.

```
AKSHSHAR-M-KODS:vagrant akshshar$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:
```

```
RP/0/RP0/CPU0:rtr1# show running-config router ospf
Mon Jul 18 17:50:47.851 UTC
router ospf apphost
 area 0
  interface Loopback0
  !
  interface GigabitEthernet0/0/0/0
   cost 30
  !
  interface GigabitEthernet0/0/0/1
   cost 20
  !
  !
  !
```

The path cost from the GigabitEthernet0/0/0/0 interface is greater than that from the GigabitEthernet0/0/0/1 interface. Hence, failover takes place to the GigabitEthernet0/0/0/1 interface for traffic from `rt1` to `rtr2`.

20. Verify the OSPF path failover on `rtr1`.

The Loopback 0 interface IP address of `rtr1` in this example is 2.2.2.2

```
RP/0/RP0/CPU0:rtr1# show route 2.2.2.2
Mon Jul 18 18:01:49.297 UTC

Routing entry for 2.2.2.2/32
  Known via "ospf apphost", distance 110, metric 21, type intra area
  Installed Jul 18 16:47:45.705 for 01:14:03
  Routing Descriptor Blocks
    11.1.1.20, from 2.2.2.2, via GigabitEthernet0/0/0/1
      Route metric is 21
  No advertising protos.
RP/0/RP0/CPU0:rtr1#
```

The next hop for `rtr1` is 11.1.1.20 through the backup reference link: GigabitEthernet0/0/0/1

You have successfully configured OSPF path failover by using iPerf and Netconf on vagrant.

Using Solenoid with exaBGP on Vagrant

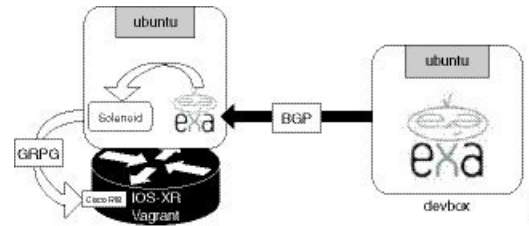
Solenoid is an application that can be used to bridge route updates between an exaBGP application instance and the RIB table on Cisco IOS XR. This section explains how you can install and use the Solenoid application with the exaBGP application for route filtering on vagrant.

Topology

The following topology is used to demonstrate the hosting of Solenoid and exaBGP applications on XR.

Two instances of Ubuntu on Vagrant are used in this topology. The vagrant instance on the right is the devbox that uses exaBGP to generate BGP route updates. The route updates are sent to the vagrant instance (on the left) running a light version of the Cisco IOS XR router image.

Figure 20: BGP Route Filtering on XR with Solenoid and exaBGP



The following workflow is used for BGP route filtering by Solenoid:

1. Solenoid receives the route updates from exaBGP.
2. Solenoid filters the required routes to create Cisco YANG data models for static routes.
3. Solenoid uses gRPC to send the data models to the RIB table on XR.

Procedure

Use the following steps to host and use Solenoid and exaBGP applications for BGP route filtering.

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.

```
lisroach@LISROACH-M-J0AY ~/W/X/S/vagrant> vagrant box list
IOS XRv          (virtualbox, 0)
```

4. Create a working directory.

```
LISROACH-M-J0AY:~ lisroach$ mkdir ~/iosxrv
LISROACH-M-J0AY:~ lisroach$ cd ~/iosxrv
```

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

```
LISROACH-M-J0AY:~ lisroach$ 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 repository containing the Solenoid application source code.

```
lisroach@LISROACH-M-J0AY ~/Workspace> git clone https://github.com/ios-xr/Solenoid.git
Cloning into 'Solenoid'...
remote: Counting objects: 1539, done.
remote: Compressing objects: 100% (623/623), done.
remote: Total 1539 (delta 884), reused 1508 (delta 866), pack-reused 0
Receiving objects: 100% (1539/1539), 713.76 KiB | 317.00 KiB/s, done.
Resolving deltas: 100% (884/884), done.
Checking connectivity... done.
lisroach@LISROACH-M-J0AY ~/Workspace>
```


7. Launch the vagrant instance running devbox.

```
lisroach@LISROACH-M-J0AY ~/Workspace> cd Solenoid/vagrant
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant up devbox
```

8. Verify if the exaBGP application is running successfully on devbox.

- a. Launch the exaBGP screen.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo screen -ls
There is a screen on:
      1762.exabgp      (09/27/2016 10:43:34 PM)      (Detached)
1 Socket in /var/run/screen/S-root.
```

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo screen -r exabgp
Tue, 27 Sep 2016 23:43:25 | INFO | 1764 | processes
| Command from process add-routes : announce route 2.2.2.0/24 next-hop self
Tue, 27 Sep 2016 23:43:25 | INFO | 1764 | reactor |
Route added to neighbor 11.1.1.10 local-ip 11.1.1.20
local-as 65000 peer-as 65000 router-id 11.1.1.20
family-allowed in-open : 2.2.2.0/24 next-hop 11.1.1.20
```

- b. Detach from the exaBGP screen without killing the processes or destroying the screen by pressing **CTRL+a,d**.

- c. Verify if the screen has been successfully detached.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo screen -r exabgp
[detached from 1762.exabgp]
```

9. Create the LXC (container) to host the Solenoid application.

- a. Access devbox through SSH.

```
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-92-generic x86_64)

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

```

- b. Install the LXC tools for creating the LXC for Solenoid.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get update
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt -y install lxc
```

- c. Create the Solenoid LXC.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-create -t ubuntu --name solenoid
```

- d. Launch the Solenoid LXC.

The password for user ubuntu is **ubuntu**.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-start --name solenoid
solenoid login: init: setvtrgb main process (428) terminated with status 1
init: plymouth-upstart-bridge main process ended, respawning
ubuntu
Password:
```

10. Install the exaBGP application dependencies inside the LXC.

```
ubuntu@solenoid:~$ sudo apt-get -y install git curl
screen python-dev python-setuptools
```

```
[sudo] password for ubuntu: ubuntu
ubuntu@solenoid:~$ sudo easy_install pip
ubuntu@solenoid:~$ sudo pip install virtualenv exabgp
```

11. Install the Solenoid application code dependencies inside the LXC.

a. Clone the Solenoid application code from github.

```
ubuntu@solenoid:~$ git clone https://github.com/ios-xr/Solenoid.git
```

b. Activate the virtual environment inside the Solenoid directory.

```
ubuntu@solenoid:~$ cd Solenoid
ubuntu@solenoid:~$ virtualenv venv
ubuntu@solenoid:~$ source venv/bin/activate
```



Note You cannot activate Solenoid without activating the virtual environment.

c. Install the Solenoid application code dependencies in the virtual environment.

```
(venv) ubuntu@solenoid:~$ pip install grpcio
(venv) ubuntu@solenoid:~$ python setup.py install
```

12. Create the configuration file for Solenoid, `solenoid.config`, in the Solenoid directory with the following contents.

```
[default] # Name you choose for the node
transport: gRPC # Either gRPC or RESTconf
ip: 11.1.1.10 # IP address of the destination RIB table (the XR device you intend
to control)
port: 57777 # Depends on what is configured for your gRPC or RESTconf servers
username: vagrant # Username for the XR device
password: vagrant # Password for the XR device
```

13. Create the configuration file for exaBGP, `router.ini`, in your home directory with the following contents.

```
group demo {
    router-id 11.1.1.10;

    process monitor-neighbors {
        encoder json;
        receive {
            parsed;
            updates;
            neighbor-changes;
        }
        run /usr/bin/env python /home/ubuntu/Solenoid/solenoid/edit_rib.py -f
'/home/ubuntu/Solenoid/filter.txt';
    }

    neighbor 11.1.1.20 {
        local-address 11.1.1.10;
        local-as 65000;
        peer-as 65000;
    }
}
```

```
    }
}
```

14. Change the SSH port inside the LXC.

```
(venv) ubuntu@solenoid:~$ sudo sed -i s/Port\ 22/Port\ 58822/ /etc/ssh/sshd_config

(venv) ubuntu@solenoid:~$ cat /etc/ssh/sshd_config | grep Port
Port 58822
```

15. Shut down the LXC.

```
(venv) ubuntu@solenoid:~$ sudo shutdown -h now
(venv) ubuntu@solenoid:~$
Broadcast message from ubuntu@solenoid
(/dev/lxc/console) at 23:00 ...

The system is going down for halt NOW!
...
```

16. Package the LXC tar ball as the root user.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo -s
root@vagrant-ubuntu-trusty-64:~# cd /var/lib/lxc/solenoid/rootfs/
root@vagrant-ubuntu-trusty-64:~# tar -czvf /vagrant/solenoid.tgz *

root@vagrant-ubuntu-trusty-64:~# exit
exit
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> pwd
/Users/lisroach/Workspace/Solenoid/vagrant
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> ls -la solenoid.tgz
-rw-r--r-- 1 lisroach staff 252417007 Aug 2 11:27 solenoid.tgz
```

17. Launch the topology by launching the vagrant instance in the Solenoid/vagrant/ directory.

```
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> pwd
/Users/lisroach/Workspace/Solenoid/vagrant

lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant up
Bringing machine 'xrv' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
...
==> xrv: Importing base box 'IOS XRv'...
==> xrv: Machine 'xrv' has a post `vagrant up` message. This is a message
==> xrv: from the creator of the Vagrantfile, and not from Vagrant itself:
==> xrv:
==> xrv:
==> xrv: Welcome to the IOS XRv (64-bit) VirtualBox.

lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant status
Current machine states:

xrv                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`.
```

18. Launch the solenoid GUI on your host machine by entering localhost:57780 in your web browser.

Figure 21: Solenoid GUI on Host Machine

Solenoid Demo on IOS-XRv

Topology



A BGP network is shown above. Networks are automatically announced and withdrawn by another bgp speaker at a fixed interval.

Solenoid is running in an Ubuntu container, that is sitting on an IOS-XRv image.

Prefix Filtering

Solenoid can use filtering to only pass designated prefixes that fall in a prefix range or individual prefix

In the website you can add or remove prefixes or a prefix range using the toggle key and input box

Prefix

Add Remove

Prefix:

Prefix List

Latest Update on ExaBGP

Solenoid is currently listening to the updates that Exabgp receives. It will then inject the announce and withdraw request into the rib table. A controller of any sort can be in place of exabgp.

Peer IP	Update Type	Next Hop	Network	Time
11.1.1.20	withdraw		3.3.3.0/24	28 Sep 2016 10:53:19
11.1.1.20	withdraw		2.2.2.0/24	28 Sep 2016 10:53:17
11.1.1.20	withdraw		1.1.1.0/24	28 Sep 2016 10:53:15
11.1.1.20	announce	11.1.1.20	3.3.3.0/24	28 Sep 2016 10:53:13
11.1.1.20	announce	11.1.1.20	2.2.2.0/24	28 Sep 2016 10:53:11
11.1.1.20	announce	11.1.1.20	1.1.1.0/24	28 Sep 2016 10:53:9
11.1.1.20	withdraw		3.3.3.0/24	28 Sep 2016 10:53:5
11.1.1.20	withdraw		2.2.2.0/24	28 Sep 2016 10:53:4

RIB table on Router

The RIB table is shown here through restconf. The RIB table is continuously updated, reflecting what Solenoid has injected.

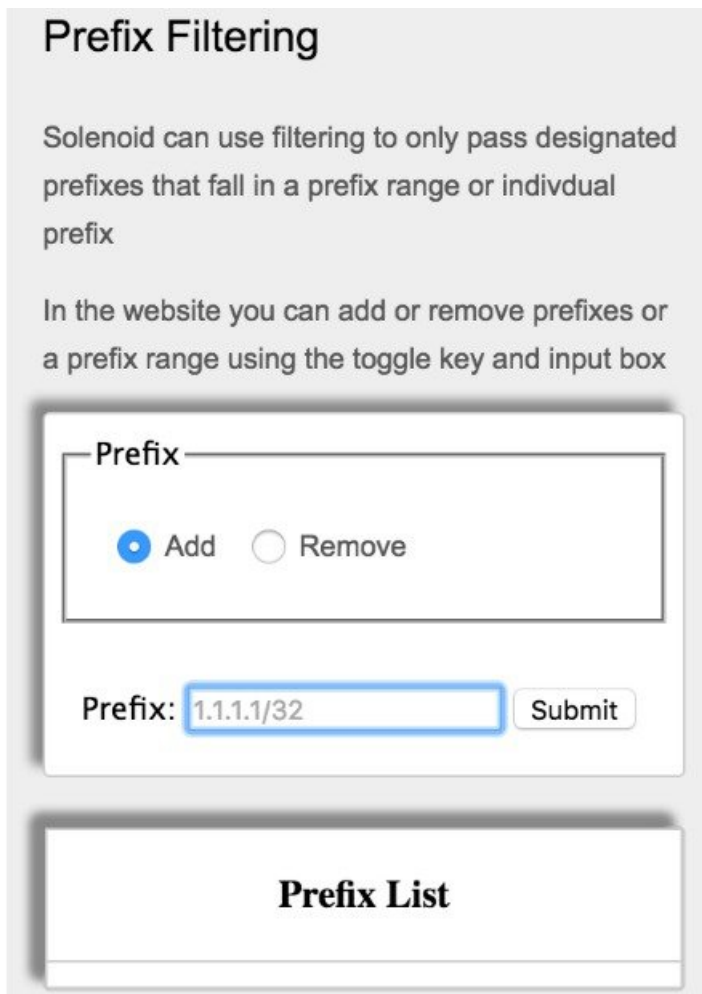
Prefix	Next Hop
0.0.0.0	10.0.2.2

365934

The GUI displays the IOS XR RIB table. You can see the exaBGP routes that were added and withdrawn by Solenoid.

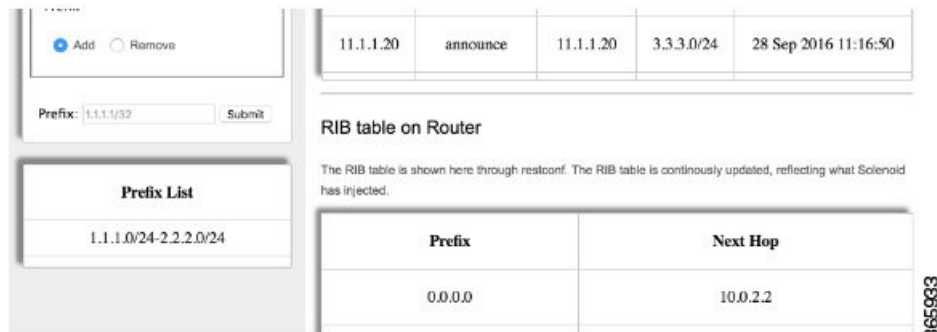
You can enable filtering of prefixes, by adding the prefixes to the filtering file as shown. The file acts as an allowed list by allowing the entered prefixes and dropping all other prefixes.

Figure 22: Prefix Filter Creation on Solenoid GUI



For the sake of illustration, we add the prefix range: 1.1.1.0/24 to 2.2.2.0/24 to the allowed list filter. Because of this configuration, the 3.3.3.0/24 prefix gets filtered out and is not added to the RIB table, as shown.

Figure 23: Prefix Filtering on Solenoid GUI



- 19. Verify the RIB table on XR.

```
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant port xrv
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)
57780 (guest) => 57780 (host)
58822 (guest) => 58822 (host)
```

```
-----
(venv) lisroach@LISROACH-M-J0AY ~/W/S/vagrant> ssh -p 2223 vagrant@localhost
vagrant@localhost's password:
```

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

```
RP/0/RP0/CPU0:ios# show ip route
Wed Sep 28 18:33:18.266 UTC
```

```
Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
U - per-user static route, o - ODR, L - local, G - DAGR, l - LISP
A - access/subscriber, a - Application route
M - mobile route, r - RPL, (!) - FRR Backup path
```

```
Gateway of last resort is 10.0.2.2 to network 0.0.0.0
```

```
S* 0.0.0.0/0 [1/0] via 10.0.2.2, 01:01:34
C 10.0.2.0/24 is directly connected, 01:03:27, MgmtEth0/RP0/CPU0/0
L 10.0.2.15/32 is directly connected, 01:03:27, MgmtEth0/RP0/CPU0/0
L 10.1.1.5/32 is directly connected, 01:01:34, Loopback1
C 11.1.1.0/24 is directly connected, 01:01:34, GigabitEthernet0/0/0/0
L 11.1.1.10/32 is directly connected, 01:01:34, GigabitEthernet0/0/0/0
RP/0/RP0/CPU0:ios#
```

Notice that the default route of 0.0.0.0 is the only static route in the RIB table, prior to running the Solenoid application on XR.

20. Open a second Git bash window in parallel and access the Solenoid LXC.

```
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> vagrant port xrv
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)
57780 (guest) => 57780 (host)
58822 (guest) => 58822 (host)
```

```
-----
lisroach@LISROACH-M-J0AY ~/W/S/vagrant> ssh -p 58822 ubuntu@localhost
The authenticity of host '[localhost]:58822 ([127.0.0.1]:58822)' can't be established.
ECDSA key fingerprint is SHA256:Swie3V2VIYDNCACaRLbsjQa7417yIM6hpbeimNwZr1o.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '[localhost]:58822' (ECDSA) to the list of known hosts.
ubuntu@localhost'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: Thu Sep 22 21:31:13 2016

ubuntu@solenoid:~$
```

21. Launch the exaBGP screen to verify that Solenoid is in operation.

```
ubuntu@solenoid:~$ screen -ls
There are screens on:
    1423.website      (09/28/2016 05:38:22 PM)      (Detached)
    1421.exabgp      (09/28/2016 05:38:22 PM)      (Detached)
2 Sockets in /var/run/screen/S-ubuntu.
ubuntu@solenoid:~$
```

```
-----
ubuntu@solenoid:~$ screen -r exabgp
Wed, 28 Sep 2016 18:35:04 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:06 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:11 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:13 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:17 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:19 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:25 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:27 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:37 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:37 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:38 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:40 | INFO | 1436 | solenoid | ANNOUNCE | OK
Wed, 28 Sep 2016 18:35:44 | INFO | 1436 | solenoid | WITHDRAW | OK
Wed, 28 Sep 2016 18:35:46 | INFO | 1436 | solenoid | WITHDRAW | OK
```

The OK messages verify that the Solenoid application is running as desired.

22. Open the first Gitbash window, and view the RIB table on XR with Solenoid in operation.

```
RP/0/RP0/CPU0:ios#show ip route
Wed Sep 28 18:49:22.165 UTC

Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
U - per-user static route, o - ODR, L - local, G - DAGR, l - LISIP
A - access/subscriber, a - Application route
M - mobile route, r - RPL, (!) - FRR Backup path

Gateway of last resort is 10.0.2.2 to network 0.0.0.0

S* 0.0.0.0/0 [1/0] via 10.0.2.2, 01:17:38
S 1.1.1.0/24 [1/0] via 11.1.1.20, 00:00:00
C 10.0.2.0/24 is directly connected, 01:19:31, MgmtEth0/RP0/CPU0/0
L 10.0.2.15/32 is directly connected, 01:19:31, MgmtEth0/RP0/CPU0/0
L 10.1.1.5/32 is directly connected, 01:17:38, Loopback1
C 11.1.1.0/24 is directly connected, 01:17:38, GigabitEthernet0/0/0/0
L 11.1.1.10/32 is directly connected, 01:17:38, GigabitEthernet0/0/0/0

-----
```

```

RP/0/RP0/CPU0:ios#show ip route
Wed Sep 28 18:49:25.660 UTC

Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
U - per-user static route, o - ODR, L - local, G - DAGR, l - LISP
A - access/subscriber, a - Application route
M - mobile route, r - RPL, (!) - FRR Backup path

Gateway of last resort is 10.0.2.2 to network 0.0.0.0

S*  0.0.0.0/0 [1/0] via 10.0.2.2, 01:17:42
S   1.1.1.0/24 [1/0] via 11.1.1.20, 00:00:03
S   2.2.2.0/24 [1/0] via 11.1.1.20, 00:00:01
C   10.0.2.0/24 is directly connected, 01:19:35, MgmtEth0/RP0/CPU0/0
L   10.0.2.15/32 is directly connected, 01:19:35, MgmtEth0/RP0/CPU0/0
L   10.1.1.5/32 is directly connected, 01:17:42, Loopback1
C   11.1.1.0/24 is directly connected, 01:17:42, GigabitEthernet0/0/0/0
L   11.1.1.10/32 is directly connected, 01:17:42, GigabitEthernet0/0/0/0
RP/0/RP0/CPU0:ios#

```

The RIB table clearly shows that because we added the 1.1.1.0/24 and 2.2.2.0/24 prefixes to the Solenoid GUI, these prefixes are added to the RIB table on XR when Solenoid is launched on XR (through exaBGP screen).

This verifies the successful hosting and operation of Solenoid application on XR for filtering BGP routes, by using exaBGP and the Solenoid GUI on vagrant.