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 1
- Using Puppet for Configuring Cisco IOS XR, on page 5
- Using Configuration Management Tools on Vagrant, on page 11

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 1: Chef Resources

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

Installing and Configuring the Chef Client

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

Prerequisites

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

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

Configuration Procedure

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

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

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

   You have entered the IOS XR prompt.

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

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

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

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


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

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

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

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

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

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

4. Install the Chef Client.

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

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

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

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

7. Run the Chef Client.

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

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

The Chef Client is successfully installed on IOS XR.

Creating a Chef Cookbook with Recipes

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

Prerequisites

Ensure the following requirements are met before you proceed:

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

Configuration Procedure

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

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

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

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

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

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

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

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

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

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

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

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

The iperf RPM is installed on IOS XR.

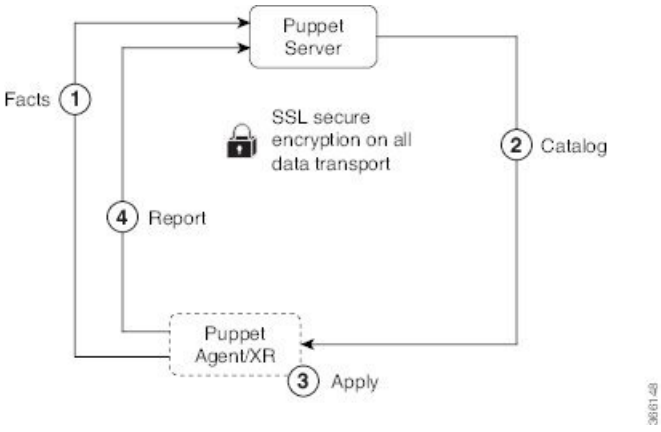
For additional details on the Chef Client, refer to https://docs.chef.io/chef_client.html

Using Puppet for Configuring Cisco IOS XR

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

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

Figure 1: 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 2: Puppet Resources

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

Installing and Configuring the Puppet Agent

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

Prerequisites

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

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

Configuration Procedure

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

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

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

You have entered the IOS XR prompt.

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

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

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

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

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

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

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

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

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

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

4. Install the Puppet Agent.

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

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

The Puppet Agent is successfully installed on IOS XR.

Creating a Puppet Manifest

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

Prerequisites

Ensure the following requirements are met before you proceed:

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

Configuration Procedure

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

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

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

```
# Manifest to demo builtin providers
#

class ciscopuppet::demo_builtin_providers {

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

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

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

```
  }

  service { $service:
    ensure => running,
  }

}

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

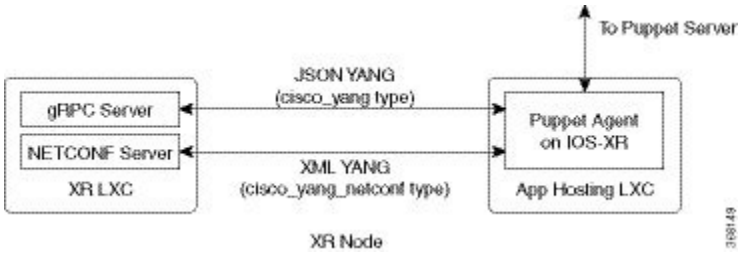
2. Access and trigger the Puppet Agent to converge the system based on the manifest defined on the Puppet Server.

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

Using Yang Models with Puppet on IOS XR

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

Figure 2: Yang with Puppet Workflow



Installing the Cisco Yang Puppet Module on Puppet Server

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

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

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

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

Sample Puppet Manifest By Using the cisco_yang Type

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

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

Sample Puppet Manifest By Using the cisco_yang_netconf Type

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

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

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

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

Using Configuration Management Tools on Vagrant

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

Pre-requisites for Using Vagrant

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

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

Using Puppet on Vagrant

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

Procedure

To start using Puppet on vagrant, use the following steps.

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

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

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

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

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

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

4. Create a working directory.

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

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

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

6. Clone the `vagrant-xrdocs` repository.

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

7. Navigate to the `vagrant-xrdocs` repository and locate the `puppet-tutorials/app_hosting/centos-pm` directory for launching the Puppet server.

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

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

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

8. Launch the vagrant instance for Puppet server.

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

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

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

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

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

9. Create and apply a sample Puppet manifest file.

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

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

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

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

Locate and copy the manifest file.

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

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

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

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

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

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

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

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

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

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

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


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

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

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

Using Ansible for Hosting Applications

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

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

Ansible Modes of Operation

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

The two modes operate differently from each other.

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

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

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

     For information on accessing TPNNS, see Accessing the Third-Party Network Namespace on Cisco IOS XR Linux Shell.
  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 3: Accessing IOS XR Through Ansible

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

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

Using Ansible On Vagrant

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

Sample Ansible Operation on XR

To start using Ansible on XR, use the following steps:

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

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

```
annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant up
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr' up with 'virtualbox' provider...
==> devbox: Checking if box 'ubuntu/trusty64' is up to date...
==> devbox: A newer version of the box 'ubuntu/trusty64' is available! You currently
==> devbox: have version '20160801.0.0'. The latest is version '20160826.0.1'. Run
==> devbox: `vagrant box update` to update.
==> devbox: Clearing any previously set forwarded ports...
==> devbox: Clearing any previously set network interfaces...
==> devbox: Preparing network interfaces based on configuration...
    devbox: Adapter 1: nat
    devbox: Adapter 2: intnet
==> devbox: Forwarding ports...
    devbox: 22 (guest) => 2222 (host) (adapter 1)
==> devbox: Running 'pre-boot' VM customizations...
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2222
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    devbox: Warning: Remote connection disconnect. Retrying...
    ...
==> devbox: Machine booted and ready!
...
    devbox: Guest Additions Version: 4.3.36
    devbox: VirtualBox Version: 5.0
==> devbox: Configuring and enabling network interfaces...
==> devbox: Mounting shared folders...
    devbox: /vagrant => C:/Users/annseque/vagrant-xrdocs/ansible-tutorials/app_hosting
==> devbox: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> devbox: flag to force provisioning. Provisioners marked to run always will still
run.
==> rtr: Clearing any previously set forwarded ports...
==> rtr: Fixed port collision for 57722 => 2222. Now on port 2200.
==> rtr: Clearing any previously set network interfaces...
==> rtr: Preparing network interfaces based on configuration...
    rtr: Adapter 1: nat
    rtr: Adapter 2: intnet
==> rtr: Forwarding ports...
    rtr: 57722 (guest) => 2200 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
    rtr: 58822 (guest) => 58822 (host) (adapter 1)
==> rtr: Running 'pre-boot' VM customizations...
==> rtr: Booting VM...
==> rtr: Waiting for machine to boot. This may take a few minutes...
    rtr: SSH address: 127.0.0.1:2200
    rtr: SSH username: vagrant
    rtr: SSH auth method: private key
    rtr: Warning: Remote connection disconnect. Retrying...
    ...
==> rtr: Machine booted and ready!
...
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: flag to force provisioning. Provisioners marked to run always will still run.

==> rtr: Machine 'rtr' has a post `vagrant up` message. This is a message
==> rtr: from the creator of the Vagrantfile, and not from Vagrant itself:
==> rtr:
==> rtr:
==> rtr:     Welcome to the IOS XRv (64-bit) Virtualbox.
```

```
==> rtr:     To connect to the XR Linux shell, use: 'vagrant ssh'.
==> rtr:     To ssh to the XR Console, use: 'vagrant port' (vagrant version > 1.8)
==> rtr:     to determine the port that maps to guestport 22,
==> rtr:     then: 'ssh vagrant@localhost -p <forwarded port>'
...


annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant status
Current machine states:

devbox                    running (virtualbox)
rtr                       running (virtualbox)

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

2. Configure access to XR (`rtr`).

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

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

    ...

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

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


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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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


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

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

        ""
    ]
}

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

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

You have successfully used Ansible on vagrant.

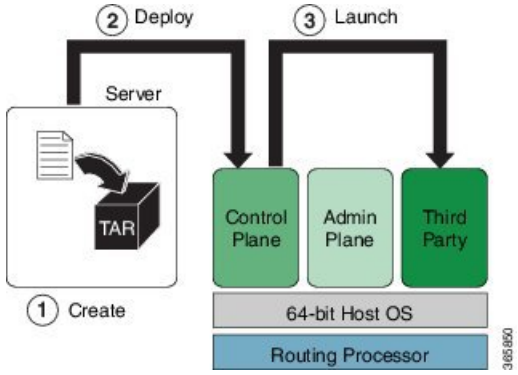
Launching a Linux Container (LXC) By Using Ansible on Vagrant

This section describes how you can launch your own container (LXC) by using Ansible on vagrant.

Workflow for Deploying Your LXC Container

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

Figure 3: 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, before proceeding with the following steps.

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

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

annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant up
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr' up with 'virtualbox' provider...
==> devbox: Checking if box 'ubuntu/trusty64' is up to date...
==> devbox: A newer version of the box 'ubuntu/trusty64' is available! You currently
==> devbox: have version '20160801.0.0'. The latest is version '20160826.0.1'. Run
```

```
==> devbox: `vagrant box update` to update.
==> devbox: Clearing any previously set forwarded ports...
==> devbox: Clearing any previously set network interfaces...
==> devbox: Preparing network interfaces based on configuration...
    devbox: Adapter 1: nat
    devbox: Adapter 2: intnet
==> devbox: Forwarding ports...
    devbox: 22 (guest) => 2222 (host) (adapter 1)
==> devbox: Running 'pre-boot' VM customizations...
==> devbox: Booting VM...
==> devbox: Waiting for machine to boot. This may take a few minutes...
    devbox: SSH address: 127.0.0.1:2222
    devbox: SSH username: vagrant
    devbox: SSH auth method: private key
    devbox: Warning: Remote connection disconnect. Retrying...
    ...
==> devbox: Machine booted and ready!
...
    devbox: Guest Additions Version: 4.3.36
    devbox: VirtualBox Version: 5.0
==> devbox: Configuring and enabling network interfaces...
==> devbox: Mounting shared folders...
   devbox: /vagrant => C:/Users/annseque/vagrant-xrdocs/ansible-tutorials/app_hosting
==> devbox: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> devbox: flag to force provisioning. Provisioners marked to run always will still
run.
==> rtr: Clearing any previously set forwarded ports...
==> rtr: Fixed port collision for 57722 => 2222. Now on port 2200.
==> rtr: Clearing any previously set network interfaces...
==> rtr: Preparing network interfaces based on configuration...
    rtr: Adapter 1: nat
    rtr: Adapter 2: intnet
==> rtr: Forwarding ports...
    rtr: 57722 (guest) => 2200 (host) (adapter 1)
    rtr: 22 (guest) => 2223 (host) (adapter 1)
    rtr: 58822 (guest) => 58822 (host) (adapter 1)
==> rtr: Running 'pre-boot' VM customizations...
==> rtr: Booting VM...
==> rtr: Waiting for machine to boot. This may take a few minutes...
    rtr: SSH address: 127.0.0.1:2200
    rtr: SSH username: vagrant
    rtr: SSH auth method: private key
    rtr: Warning: Remote connection disconnect. Retrying...
    ...
==> rtr: Machine booted and ready!
...
==> rtr: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> rtr: flag to force provisioning. Provisioners marked to run always will still run.

==> rtr: Machine 'rtr' has a post `vagrant up` message. This is a message
==> rtr: from the creator of the Vagrantfile, and not from Vagrant itself:
==> rtr:
==> rtr:
==> rtr:     Welcome to the IOS XRv (64-bit) Virtualbox.
==> rtr:     To connect to the XR Linux shell, use: 'vagrant ssh'.
==> rtr:     To ssh to the XR Console, use: 'vagrant port' (vagrant version > 1.8)
==> rtr:     to determine the port that maps to guestport 22,
==> rtr:     then: 'ssh vagrant@localhost -p <forwarded port>'
...


annseque@ANNSEQUE-WS02 MINGW64 ~/vagrant-xrdocs/ansible-tutorials/app_hosting (master)
$ vagrant status
Current machine states:
```

```
devbox                    running (virtualbox)
rtr                       running (virtualbox)

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

2. Configure access to XR (`rtr`), if not done already.

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

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

 ...

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

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


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

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

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

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

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

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

3. Access the `devbox` through SSH, and install LXC tools.

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

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

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

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

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

 ...

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

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


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

--------------------------------------------------------------------------------------------
vagrant@vagrant-ubuntu-trusty-64:~$ sudo apt-get -y install lxc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bridge-utils cgmanager cloud-image-utils debootstrap dnsmasq-base euca2ools
  genisoimage libaio1 libboost-system1.54.0 libboost-thread1.54.0 liblxc1
  libmnl0 libnetfilter-conntrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbd1 libseccomp2 libxslt1.1 lxc-templates python-distro-info python-lxml
  python-requestbuilder python-setuptools python3-lxc qemu-utils sharutils
  uidmap
Suggested packages:
  cgmanager-utils wodim cdrkit-doc btrfs-tools lvm2 lxctl qemu-user-static
  python-lxml-dbg bsd-mailx mailx
The following NEW packages will be installed:
  bridge-utils cgmanager cloud-image-utils debootstrap dnsmasq-base euca2ools
  genisoimage libaio1 libboost-system1.54.0 libboost-thread1.54.0 liblxc1
  libmnl0 libnetfilter-conntrack3 libnspr4 libnss3 libnss3-nssdb librados2
  librbd1 libseccomp2 libxslt1.1 lxc lxc-templates python-distro-info
  python-lxml python-requestbuilder python-setuptools python3-lxc qemu-utils
  sharutils uidmap
0 upgraded, 30 newly installed, 0 to remove and 52 not upgraded.
Need to get 6,469 kB of archives.
After this operation, 25.5 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main libaio1 amd64 0.3.109-4 [6,364 B]
...
Get:30 http://archive.ubuntu.com/ubuntu/ trusty-updates/main debootstrap all
1.0.59ubuntu0.5 [29.6 kB]
Fetched 6,469 kB in 22s (289 kB/s)
Selecting previously unselected package libaio1:amd64.
(Reading database ... 62989 files and directories currently installed.)
Preparing to unpack .../libaio1_0.3.109-4_amd64.deb ...
...
Setting up lxc (1.0.8-0ubuntu0.3) ...
lxc start/running
Setting up lxc dnsmasq configuration.
```

```
Processing triggers for ureadahead (0.100.0-16) ...
Setting up lxc-templates (1.0.8-0ubuntu0.3) ...
Setting up libnss3-nssdb (2:3.23-0ubuntu0.14.04.1) ...
Setting up libnss3:amd64 (2:3.23-0ubuntu0.14.04.1) ...
Setting up librados2 (0.80.11-0ubuntu1.14.04.1) ...
Setting up librbd1 (0.80.11-0ubuntu1.14.04.1) ...
Setting up qemu-utils (2.0.0+dfsg-2ubuntu1.27) ...
Setting up cloud-image-utils (0.27-0ubuntu9.2) ...
Processing triggers for libc-bin (2.19-0ubuntu6.9) ...

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

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

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

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

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


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

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

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

6. Start the LXC container.

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

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

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

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

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

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

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

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

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

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

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

8. Shut the container down.

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

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

9. Assume the root user role.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

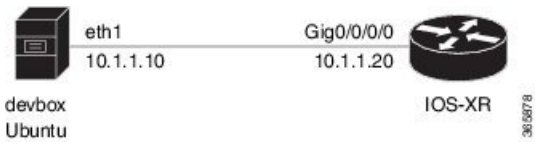
Using Netmiko and Napalm on Vagrant

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

Topology

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

Figure 4: 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.