Deployment of IPI on BM using the Ansible Playbook

Deployment Integration Team

Download the PDF version of this document or visit https://openshift-kni.github.io/baremetal-deploy/

Chapter 1. Introduction

This write-up will guide you through the process of using the Ansible playbooks to deploy a Baremetal Installer Provisioned Infrastructure (IPI) of Red Hat OpenShift 4.

For the manual details, visit our Deployment Guide

Chapter 2. Prerequisites

- Best Practice Minimum Setup: 6 Physical servers (1 provision node, 3 master and 2 worker nodes)
- Best Practice Minimum Setup for disconnected environments: 7 Physical servers (1 provision node, 1 registry node[1], 3 master and 2 worker nodes)
- Minimum Setup: 4 Physical servers (1 provision node, 3 master nodes)
- Minimum Setup for disconnected environments: 5 Physical servers (1 provision node, 1 registry node[1], 3 master nodes)
- Each server needs 2 NICs pre-configured. NIC1 for the private network and NIC2 for the baremetal network. NIC interface names must be identical across all nodes[2]
- It is recommended each server have a RAID-1 configured and initialized (though not enforced)
- Each server must have IPMI configured
- Each server must have DHCP setup for the baremetal NICs
- Each server must have DNS setup for the API, wildcard applications
- A DNS VIP is IP on the `baremetal` network is required for reservation. Reservation is done via our DHCP server (though not required).
- Optional - Include DNS entries for the hostnames for each of the servers
- Download a copy of your Pull Secret

Due to the complexities of properly configuring an environment, it is recommended to review the following steps prior to running the Ansible playbook as without proper setup, the Ansible playbook won't work.

The section to review and ensure proper configuration are as follows:

- Validation checklist for nodes
- One of the Create DNS records sections
  - Create DNS records on a DNS server (Option 1)
  - Create DNS records using dnsmasq (Option 2)
- One of the Create DHCP reservation sections
  - Create DHCP reservations (Option 1)
  - Create DHCP reservations using dnsmasq (Option 2)
- An existing Registry node (details on creating a registry if required below)
  - Create a disconnected registry
- An existing webserver to cache required files and the RHCOS images (details on creating a webserver if required below)
  - Webserver

Once the above is complete, install Red Hat Enterprise Linux (RHEL) 8.x on your provision node

and create a user (i.e. `kni`) to deploy as non-root and provide that user `sudo` privileges.

For simplicity, the steps to create the user named `kni` is as follows:

1. Login into the provision node via `ssh`
2. Create a user (i.e `kni`) to deploy as non-root and provide that user `sudo` privileges

```
useradd kni
passwd kni
echo "kni ALL=(root) NOPASSWD:ALL" | tee -a /etc/sudoers.d/kni
chmod 0440 /etc/sudoers.d/kni
```

3. Enable a `dnf` local repository on the provision host
4. Manually install `python3-crypto` and `python3-pyghmi` packages on the provision host

[1] If creating the mirrored registry, this system will require online access. The registry node may be a virtual machine in order to reduce physical server footprint.

[2] https://github.com/openshift/installer/issues/2762

Chapter 3. Tour of the Ansible Playbook

- `inventory` - contains the file `hosts.sample` that:
  - contains all the modifiable variables, their default values, and their definition. Some variables are empty ensuring users give an explicit value.
  - the setting up of your provision node, master nodes, and worker nodes. Each section will require additional details (i.e. Management credentials).
- `roles` - contains two roles: `node-prep` and `installer`. `node-prep` handles all the prerequisites that the provisioner node requires prior to running the installer. The `installer` role handles extracting the installer, setting up the manifests, and running the Red Hat OpenShift installation.

The tree structure is shown below:

```
├──── ansible.cfg
├──── inventory
│     └──── hosts.sample
├──── playbook.yml
└──── roles
    ├──── installer
    │     ├──── defaults
    │     │     └──── main.yml
    │     ├──── files
    │     ├──── handlers
    │     │     └──── main.yml
    │     ├──── meta
    │     │     └──── main.yml
    │     ├──── tasks
    │     │     ├──── 10_get_oc.yml
    │     │     ├──── 15_disconnected_registry_create.yml
    │     │     ├──── 15_disconnected_registry_existing.yml
    │     │     ├──── 20_extract_installer.yml
    │     │     ├──── 23_rhcos_image_paths.yml
    │     │     ├──── 24_rhcos_image_cache.yml
    │     │     ├──── 25_create-install-config.yml
    │     │     ├──── 30_create_metal3.yml
    │     │     ├──── 40_create_manifest.yml
    │     │     ├──── 50_extramanifests.yml
    │     │     ├──── 55_customize_filesystem.yml
    │     │     ├──── 59_cleanup_bootstrap.yml
    │     │     ├──── 60_deploy_ocp.yml
    │     │     ├──── 70_cleanup_sub_man_registeration.yml
    │     │     └──── main.yml
    │     ├──── templates
    │     │     ├──── 99-etc-chrony.conf.j2
    │     │     ├──── chrony.conf.j2
    │     │     ├──── install-config-appends.j2
    │     │     ├──── install-config.j2
```

```
│   │   └── metal3-config.j2
│   ├── tests
│   │   ├── inventory
│   │   └── test.yml
│   └── vars
│       └── main.yml
└── node-prep
    ├── defaults
    │   └── main.yml
    ├── handlers
    │   └── main.yml
    ├── library
    │   └── nmcli.py
    ├── meta
    │   └── main.yml
    ├── tasks
    ├── 100_power_off_cluster_servers.yml
    ├── 10_validation.yml
    │   ├── 15_validation_disconnected_registry.yml
    │   ├── 20_sub_man_register.yml
    │   ├── 30_req_packages.yml
    │   ├── 40_bridge.yml
    │   ├── 45_networking_facts.yml
    │   ├── 50_modify_sudo_user.yml
    │   ├── 60_enabled_services.yml
    │   ├── 70_enabled_fw_services.yml
    │   ├── 80_libvirt_pool.yml
    │   ├── 90_create_config_install_dirs.yml
    │   └── main.yml
    ├── templates
    │   ├── dir.xml.j2
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml
```

Chapter 4. Running the Ansible Playbook

The following are the steps to successfully run the Ansible playbook.

4.1. `git` clone the Ansible playbook

The first step to using the Ansible playbook is to clone the baremetal-deploy repository.

This should be done on a system that can access the provision host

1. Clone the `git` repository

```
[user@laptop ~]$ git clone https://github.com/openshift-kni/baremetal-deploy.git
```

Ensure `git` is installed on your localhost

2. Change to the `ansible-ipi-install` directory

```
[user@laptop ~]$ cd /path/to/git/repo/baremetal-deploy/ansible-ipi-install
```

4.2. The `ansible.cfg` file

While the `ansible.cfg` may vary upon your environment a sample is provided in the repository.

```
[defaults]
inventory=./inventory
remote_user=kni
callback_whitelist = profile_tasks

[privilege_escalation]
become_method=sudo
```

Ensure to change the `remote_user` as deemed appropriate for your environment. The `remote_user` is the user previously created on the provision node.

4.3. Ansible version

Ensure that your environment is using Ansible 2.9 or greater. The following command can be used to verify.

```
ansible --version
ansible 2.9.1
  config file = /path/to/baremetal-deploy/ansible-ipi-install/ansible.cfg
  configured module search path = ['/path/to/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.7.2 (default, Jan 16 2019, 19:49:22) [GCC 8.2.1 20181215 (Red Hat
8.2.1-6)]
```

The config file section should point to the path of your `ansible.cfg`

4.4. Copy local SSH key to provision node

With the `ansible.cfg` file in place, the next step is to ensure to copy your public `ssh` key to your provision node using `ssh-copy-id`.

From the system that is to run the playbook,

```
$ ssh-copy-id <user>@provisioner.example.com
```

<user> should be the user previously created on the provision node (i.e. `kni`)

4.5. Modifying the `inventory/hosts`

The hosts file provides all the definable variables and provides a description of each variable. Some of the variables are explicitly left empty and **require** user input for the playbook to run.

The hosts file ensures all your nodes that will be used to deploy IPI on baremetal are setup. There are 4 groups: `masters`, `workers`, `provisioner`, and `registry_host` (optional). The `masters` and `workers` group collects information about the host such as its name, role, user management (i.e. iDRAC) user, user management (i.e. iDRAC) password, `ipmi_address`, `ipmi_port` to access the server and the provision mac address (NIC1) that resides on the provisioning network.

Below is a sample of the inventory/hosts file

```
[all:vars]

###############################################################################
# Required configuration variables for IPI on Baremetal Installations         #
###############################################################################

# The provisioning NIC (NIC1) used on all baremetal nodes
prov_nic=eno1
```

```
# The public NIC (NIC2) used on all baremetal nodes
pub_nic=eno2

# (Optional) Set the provisioning bridge name. Default value is 'provisioning'.
#provisioning_bridge=provisioning

# (Optional) Set the baremetal bridge name. Default value is 'baremetal'.
#baremetal_bridge=baremetal

# (Optional) Activation-key for proper setup of subscription-manager, empty value
skips registration
#activation_key=""

# (Optional) Activation-key org_id for proper setup of subscription-manager, empty
value skips registration
#org_id=""

# The directory used to store the cluster configuration files (install-config.yaml,
pull-secret.txt, metal3-config.yaml)
dir="{{ ansible_user_dir }}/clusterconfigs"

# The version of the openshift-installer, undefined or empty results in the playbook
failing with error message.
# Values accepted: 'latest-4.3', 'latest-4.4', explicit version i.e. 4.3.0-0.nightly-
2019-12-09-035405
version=""

# Enter whether the build should use 'dev' (nightly builds) or 'ga' for Generally
Available version of OpenShift
# Empty value results in playbook failing with error message.
build=""

# Provisioning IP address (default value)
prov_ip=172.22.0.3

# (Optional) Enable playbook to pre-download RHCOS images prior to cluster deployment
and use them as a local
# cache.  Default is false.
#cache_enabled=True

# (Optional) Enable IPv6 addressing instead of IPv4 addressing
#ipv6_enabled=True

# (Optional) When ipv6_enabled is set to True, but want IPv4 addressing on
provisioning network
# Default is false.
#ipv4_provisioning=True

# (Optional) When ipv6_enabled is set to True, but want IPv4 addressing on baremetal
network
#ipv4_baremetal=True
```

```
# (Optional) A list of clock servers to be used in chrony by the masters and workers
#clock_servers=["pool.ntp.org","clock.redhat.com"]

# (Optional) Provide HTTP proxy settings
#http_proxy=http://USERNAME:PASSWORD@proxy.example.com:8080

# (Optional) Provide HTTPS proxy settings
#https_proxy=https://USERNAME:PASSWORD@proxy.example.com:8080

# (Optional) comma-separated list of hosts, IP Addresses, or IP ranges in CIDR format
# excluded from proxying
# NOTE: OpenShift does not accept '*' as a wildcard attached to a domain suffix
# i.e. *.example.com
# Use '.' as the wildcard for a domain suffix as shown in the example below.
# i.e. .example.com
#no_proxy_list="172.22.0.0/24,.example.com"

# The default installer timeouts for the bootstrap and install processes may be too
short for some baremetal
# deployments. The variables below can be used to extend those timeouts.

# (Optional) Increase bootstrap process timeout by N iterations.
#increase_bootstrap_timeout=2

# (Optional) Increase install process timeout by N iterations.
#increase_install_timeout=2

# (Optional) Disable RedFish inspection to intelligently choose between IPMI or
RedFish protocol.
# By default this feature is enabled and set to true. Uncomment below to disable and
use IPMI.
#redfish_inspection=false

########################################
# Vars regarding install-config.yaml #
########################################

# Base domain, i.e. example.com
domain=""
# Name of the cluster, i.e. openshift
cluster=""
# The public CIDR address, i.e. 10.1.1.0/21
extcidrnet=""
# An IP reserved on the baremetal network.
dnsvip=""
# An IP reserved on the baremetal network for the API endpoint.
# (Optional) If not set, a DNS lookup verifies that api.<clustername>.<domain>
provides an IP
#apivip=""
# An IP reserved on the baremetal network for the Ingress endpoint.
```

```
# (Optional) If not set, a DNS lookup verifies that *.apps.<clustername>.<domain>
provides an IP
#ingressvip=""
# The master hosts provisioning nic
# (Optional) If not set, the prov_nic will be used
#masters_prov_nic=""
# Network Type (OpenShiftSDN or OVNKubernetes). Playbook defaults to OVNKubernetes.
# Uncomment below for OpenShiftSDN
#network_type="OpenShiftSDN"
# (Optional) A URL to override the default operating system image for the bootstrap
node.
# The URL must contain a sha256 hash of the image.
# See
https://github.com/openshift/installer/blob/master/docs/user/metal/customization_ipi.m
d
#   Example https://mirror.example.com/images/qemu.qcow2.gz?sha256=a07bd...
#bootstraposimage=""
# (Optional) A URL to override the default operating system image for the cluster
nodes.
# The URL must contain a sha256 hash of the image.
# See
https://github.com/openshift/installer/blob/master/docs/user/metal/customization_ipi.m
d
# Example https://mirror.example.com/images/metal.qcow2.gz?sha256=3b5a8...
#clusterosimage=""
# A copy of your pullsecret from
https://cloud.redhat.com/openshift/install/metal/user-provisioned
pullsecret=""

# (Optional) Disable BMC Certification Validation. When using self-signed certificates
for your BMC, ensure to set to True.
# Default value is False.
#disable_bmc_certificate_verification=True

# (Optional) Enable RedFish VirtualMedia/iDRAC VirtualMedia
#enable_virtualmedia=True

# (Required when enable_virtualmedia is set to True) Set an available IP address from
the baremetal net for these two variables
#provisioningHostIP=<baremetal_net_IP1>
#bootstrapProvisioningIP=<baremetal_net_IP2>

# Master nodes
# The hardware_profile is used by the baremetal operator to match the hardware
discovered on the host
# See https://github.com/metal3-io/baremetal-
operator/blob/master/docs/api.md#baremetalhost-status
# ipmi_port is optional for each host. 623 is the common default used if omitted
# poweroff is optional. True or ommited (by default) indicates the playbook will power
off the node before deploying OCP
#  otherwise set it to false
```

```
# (Optional) OpenShift 4.6+, Set Root Device Hints to choose the proper device to
install operating system on OpenShift nodes.
# root device hint options include:
['deviceName','hctl','model','vendor','serialNumber','minSizeGigabytes','wwn','rotatio
nal']
# Root Device Hint values are case sensitive.
# root_device_hint="deviceName"
# root_device_hint_value="/dev/sda"

[masters]
master-0 name=master-0 role=master ipmi_user=admin ipmi_password=password
ipmi_address=192.168.1.1 ipmi_port=623 provision_mac=ec:f4:bb:da:0c:58
hardware_profile=default poweroff=true
master-1 name=master-1 role=master ipmi_user=admin ipmi_password=password
ipmi_address=192.168.1.2 ipmi_port=623 provision_mac=ec:f4:bb:da:32:88
hardware_profile=default poweroff=true
master-2 name=master-2 role=master ipmi_user=admin ipmi_password=password
ipmi_address=192.168.1.3 ipmi_port=623 provision_mac=ec:f4:bb:da:0d:98
hardware_profile=default poweroff=true

# Worker nodes
[workers]
worker-0 name=worker-0 role=worker ipmi_user=admin ipmi_password=password
ipmi_address=192.168.1.4 ipmi_port=623 provision_mac=ec:f4:bb:da:0c:18
hardware_profile=unknown poweroff=true
worker-1 name=worker-1 role=worker ipmi_user=admin ipmi_password=password
ipmi_address=192.168.1.5 ipmi_port=623 provision_mac=ec:f4:bb:da:32:28
hardware_profile=unknown poweroff=true

# Provision Host
[provisioner]
provisioner.example.com

# Registry Host
#   Define a host here to create or use a local copy of the installation registry
#   Used for disconnected installation
# [registry_host]
# registry.example.com

# [registry_host:vars]
# The following cert_* variables are needed to create the certificates
#   when creating a disconnected registry. They are not needed to use
#   an existing disconnected registry.
# cert_country=US #it must be two letters country
# cert_state=MyState
# cert_locality=MyCity
# cert_organization=MyCompany
# cert_organizational_unit=MyDepartment

# The port exposed on the disconnected registry host can be changed from
# the default 5000 to something else by changing the following variable.
```

```
# registry_port=5000

# The directory the mirrored registry files are written to can be modified from teh
default /opt/registry by changing the following variable.
# registry_dir="/opt/registry"

# The following two variables must be set to use an existing disconnected registry.
#
# Specify a file that contains extra auth tokens to include in the
#   pull-secret if they are not already there.
# disconnected_registry_auths_file=/path/to/registry-auths.json

# Specify a file that contains the addition trust bundle and image
#   content sources for the local registry. The contents of this file
#   will be appended to the install-config.yml file.
# disconnected_registry_mirrors_file=/path/to/install-config-appends.json
```

The `ipmi_address` can take a fully qualified name assuming it is resolvable.

The `ipmi_port` examples above show how a user can specify a different `ipmi_port` for each host within their inventory file. If the `ipmi_port` variable is omitted from the inventory file, the default of 623 will be used.

A detailed description of the `vars` under the section `Vars regarding install-config.yaml` may be reviewed within Configuration Files if unsure how to populate.

4.6. The Ansible `playbook.yml`

The Ansible playbook connects to your provision host and runs through the `node-prep` role and the `installer` role. No modification is necessary. All modifications of variables may be done within the `inventory/hosts` file. A sample file is located in this repository under `inventory/hosts.sample`. From the system that is to run the playbook,

Sample `playbook.yml`

```
---
- name: IPI on Baremetal Installation Playbook
  hosts: provisioner
  roles:
    - node-prep
    - installer
```

4.7. Customizing the Node Filesystems

If you need to modify files on the node filesystems, you can augment the "fake" roots for the masters and workers under the `roles/installer/files/customize_filesystem/{master,worker}` directories. Any files added here will be included in the ignition config files for each of the machine types, leading to permanent changes to the node filesystem.

Do not place any files directly in the "fake" root — only in subdirectories. Files in the root will cause the ignition process to fail. (There is a task in the playbook to cleanup the `.gitkeep` file in the root, if it is left in place.)

This will utilize the Ignition `filetranspiler` tool, which you can read about for more information on how to use the "fake" root directories.

An example of using this customization is to disable a network interface that you need to not receive a DHCP assignment that is outside of the cluster configuration. To do this for the `eno1` interface on the master nodes, create the appropriate `etc/sysconfig/network-scripts/ifcfg-eno1` file in the "fake" root:

```
IFCFG_DIR="roles/installer/files/customize_filesystem/master/etc/sysconfig/network-
scripts"
IFNAME="eno1"
mkdir -p $IFCFG_DIR
cat << EOF > $IFCFG_DIR/ifcfg-${IFNAME}
DEVICE=${IFNAME}
BOOTPROTO=none
ONBOOT=no
EOF
```

By default these directories are empty, and the `worker` subdirectory is a symbolic link to the `master` subdirectory so that changes are universal.

4.8. Adding Extra Configurations to the OpenShift Installer

Prior to the installation of Red Hat OpenShift, you may want to include additional configuration files to be included during the installation. The `installer` role handles this.

In order to include the `extraconfigs`, ensure to place your `yaml` files within the `roles/installer/files/manifests` directory. All the files provided here will be included when the OpenShift manifests are created.

By default this directory is empty.

4.9. Pre-caching RHCOS Images

If you wish to set up a local cache of RHCOS images on your provisioning host, set the `cache_enabled` variable to `True` in your hosts file. When requested, the playbook will pre-download RHCOS images prior to actual cluster deployment.

It places these images in an Apache web server container on the provisioning host and modifies `install-config.yaml` to instruct the bootstrap VM to download the images from that web server during deployment.

If you set the `clusterosimage` and `bootstraposimage` variables, then `cache_enabled` will automatically be set to `False`. Setting these variables leaves the responsibility to the end user in ensuring the RHCOS images are readily available and accessible to the provision host.

4.10. Disconnected Registry

A disconnected registry can be used to deploy the cluster. This registry can exist or can be created.

To use a disconnected registry, set the registries host name in the `[registry_host]` group in the inventory file.

4.10.1. Creating a new disconnected registry

To create a new disconnected registry, the `disconnected_registry_auths_file` and `disconnected_registry_mirrors_file` variables must not be set.

The certificate information used to generate the host certificate must be defined. These variables must be defined as variables to the `registry_host` group in the inventory file.

```
[registry_host:vars]
cert_country=US
cert_state=MyState
cert_locality=MyCity
cert_organization=MyCompany
cert_organizational_unit=MyDepartment
```

`cert_country` must be only two letters, i.e. `US`

4.10.2. Using an Existing Registry

If no existing registry is already existing for your fully disconnected environment, visit Creating a New Disconnected Registry section.

When using an existing registry, two variables labeled `disconnected_registry_auths_file` and the `disconnected_registry_mirrors_file` must be set. These variables are located within your inventory/hosts file and the inventory/hosts.sample file can be used as reference.

The `disconnected_registry_auths_file` variable should point to a file containing json data regarding your registry information. This will be appended to the `auths` section of the pull secret by the Ansible playbook itself.

An example of the contents of the `disconnected_registry_auths_file` is shown below.

```
cat /path/to/registry-auths.json
{"registry.example.com:5000": {"auth": "ZHVtbXk6ZHsFVtbXk=", "email":
"user@example.com" } }
```

The auth password given base64 encoding of the http credentials used to create the htpasswd file.

Example:

[user@registry ~]$ b64auth=$(echo -n '<username>:<passwd>' | openssl base64)
[user@registry ~]$ echo $b64auth

The `disconnected_registry_mirrors_file` variable should point to a file containing the `additionalTrustBundle` and `imageContentSources` for the disconnected registry. The certificate that goes within the additional trust bundle is the disconnected registry node's certificate. The `imageContentSources` adds the mirrored information of the registry. The below content from the `install-config-appends.yml` file gets automatically appended by the Ansible playbook.

```
cat /path/to/install-config-appends.yml
additionalTrustBundle: |
  -----BEGIN CERTIFICATE-----
  MIIGPDCCBCSgAwIBAgIUWr1DxDq53hrsk6XVLRXUjfF9m+swDQYJKoZIhvcNAQEL
  BQAwgZAxCzAJBgNVBAYTAlVTMRAwDgYDVQQIDAdNeVN0YXRlMQ8wDQYDVQQHDAZN
  eUNpdHkxEjAQBgNVBAoMCU15Q29tcGFueTEVMBMGA1UECwwMTXlEZXBhcnRtZW50
  .
  . [ABBREVIATED CERTIFICATE FOR BREVITY]
  .
  MTMwMQYDVQQDDCpyZWdpc3RyeS5rbmk3LmNsb3VkLmxhYi5lbmcuYm9zLnJlZGhh
  dC5jb20wHhcNMjAwNDA3MjM1MzI2WhcNMzAwNDA1MjM1MzI2WjCBkDELMAkGA1UE
  -----END CERTIFICATE-----

imageContentSources:
- mirrors:
  - registry.example.com:5000/ocp4/openshift4
  source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
- mirrors:
  - registry.example.com:5000/ocp4/openshift4
  source: registry.svc.ci.openshift.org/ocp/release
- mirrors:
  - registry.example.com:5000/ocp4/openshift4
  source: quay.io/openshift-release-dev/ocp-release
```

Indentation is important in the yml file. Ensure your copy of the `install-config-appends.yml` is properly indented as in the example above.

4.11. Running the `playbook.yml`

With the `playbook.yml` set and in-place, run the `playbook.yml`

```
$ ansible-playbook -i inventory/hosts playbook.yml
```

Chapter 5. Verifying Installation

Once the playbook has successfully completed, verify that your environment is up and running.

1. Log into the provision node

```
ssh kni@provisioner.example.com
```

`kni` user is my privileged user.

2. Export the `kubeconfig` file located in the `~/clusterconfigs/auth` directory

```
export KUBECONFIG=~/clusterconfigs/auth/kubeconfig
```

3. Verify the nodes in the OpenShift cluster

```
[kni@worker-0 ~]$ oc get nodes
NAME                                 STATUS   ROLES           AGE   VERSION
master-0.openshift.example.com       Ready    master          19h   v1.16.2
master-1.openshift.example.com       Ready    master          19h   v1.16.2
master-2.openshift.example.com       Ready    master          19h   v1.16.2
worker-0.openshift.example.com       Ready    worker          19h   v1.16.2
worker-1.openshift.example.com       Ready    worker          19h   v1.16.2
```

Chapter 6. Troubleshooting

The following section troubleshoots common errors that may arise when running the Ansible playbook.

6.1. Unreachable Host

One of the most common errors is not being able to reach the `provisioner` host and seeing an error similar to

```
$ ansible-playbook -i inventory/hosts playbook.yml

PLAY [IPI on Baremetal Installation Playbook] ***********************************

TASK [Gathering Facts] **********************************************************
fatal: [provisioner.example.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to
connect to the host via ssh: ssh: Could not resolve hostname provisioner.example.com:
Name or service not known", "unreachable": true}

PLAY RECAP **********************************************************************
provisioner.example.com    : ok=0    changed=0    unreachable=1    failed=0
skipped=0    rescued=0    ignored=0
```

In order to solve this issue, ensure your `provisioner` hostname is pingable.

1. The system you are currently on can `ping` the `provisioner.example.com`

   ```
   ping provisioner.example.com
   ```

2. Once pingable, ensure that you have copied your public SSH key from your local system to the privileged user via the `ssh-copy-id` command.

   ```
   ssh-copy-id kni@provisioner.example.com
   ```

   

   When prompted, enter the password of your privileged user (i.e. `kni`).

3. Verify connectivity using the `ping` module in Ansible

```
ansible -i inventory/hosts provisioner -m ping
provisioner.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

4. Re-run the Ansible playbook

```
$ ansible-playbook -i inventory/hosts playbook.yml
```

6.2. Permission Denied Trying To Connect To Host

Another very common error is getting a permission denied error similar to:

```
$ ansible-playbook -i inventory/hosts playbook.yml

PLAY [IPI on Baremetal Installation Playbook]
*********************************************************************************************
***************

TASK [Gathering Facts]
*********************************************************************************************
*************************************
fatal: [provisioner.example.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to
connect to the host via ssh: rlopez@provisioner.example.com: Permission denied
(publickey,gssapi-keyex,gssapi-with-mic,password).", "unreachable": true}

PLAY RECAP
*********************************************************************************************
************************************************
provisioner.example.com : ok=0    changed=0    unreachable=1    failed=0    skipped=0
rescued=0    ignored=0
```

The above issue is typically related to a problem with your `ansible.cfg` file. Either it does not exist, has errors inside it, or you have not copied your SSH public key onto the provisioner.example.com system. If you notice closely, the Ansible playbook attempted to use my `rlopez` user instead of my `kni` user since my local `ansible.cfg` did not exist **AND** I had not yet set the `remote_user` parameter to `kni` (my privileged user).

1. When working with the Ansible playbook ensure you have an `ansible.cfg` located in the same directory as your `playbook.yml` file. The contents of the `ansible.cfg` should look similar to the below with the exception of changing your inventory path (location of `inventory` directory) and potentially your privileged user if not using `kni`.

```
$ cat ansible.cfg
[defaults]
inventory=/path/to/baremetal-deploy/ansible-ipi-install/inventory
remote_user=kni

[privilege_escalation]
become=true
become_method=sudo
```

2. Next, ensure that you have copied your public SSH key from your local system to the privileged user via the `ssh-copy-id` command.

```
ssh-copy-id kni@provisioner.example.com
```

When prompted, enter the password of your privileged user (i.e. `kni`).

3. Verify connectivity using the `ping` module in Ansible

```
ansible -i inventory/hosts provisioner -m ping
provisioner.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

4. Re-run the Ansible playbook

```
$ ansible-playbook -i inventory/hosts playbook.yml
```

6.3. Dig lookup requires the python 'dnspython' library and it is not installed

One of the tasks in the `node-prep` role captures your API VIP and the Ingress VIP of your environment using a `lookup` via `dig`. It does this DNS query using the `dnspython` library. This error is a little deceiving because the `dnspython` package does **not need to be installed on the remote server** (i.e. `provisioner.example.com`) but the package must be **installed on your local host** that is running the Ansible playbook.

```
TASK [node-prep : fail]
*********************************************************************************************
**********************
skipping: [provisioner.example.com]

TASK [node-prep : Verify DNS records for API VIP, Wildcard (Ingress) VIP]
***********************************************************
fatal: [provisioner.example.com]: FAILED! => {"msg": "An unhandled exception occurred
while running the lookup plugin 'dig'. Error was a <class
'ansible.errors.AnsibleError'>, original message: The dig lookup requires the python
'dnspython' library and it is not installed"}

PLAY RECAP
*********************************************************************************************
*********************************
provisioner.example.com : ok=2    changed=0    unreachable=0    failed=1    skipped=3
rescued=0    ignored=0
```

The above issue can be fixed by simply installing `python3-dns` on your local system (assuming your using an OS such as Fedora, Red Hat)

On a local host running Red Hat 8.x, run:

```
# sudo dnf install python3-dns
```

On a local host running Red Hat 7.x, run:

```
# sudo yum install python2-dns
```

On a local host running Fedora, run:

```
# sudo dnf install python3-dns
```

Re-run the Ansible playbook

```
$ ansible-playbook -i inventory/hosts playbook.yml
```

6.4. Missing python `netaddr` library

The Ansible playbook takes advantage of certain filters such as the `ipaddr` filter. In order to use this filter, your localhost running the Ansible playbook requires the python `netaddr` library.

The error when running the playbook looks like the following:

```
TASK [node-prep : Fail if Python modules are missing]
*****************************************************************************
Tuesday 05 May 2020  19:30:19 +0000 (0:00:00.512)       0:00:13.829 ***********
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Missing python module(s)
['netaddr'] on localhost\n"}
```

The above issue can be fixed by simply installing `python3-netaddr` on your local system (assuming your using an OS such as Fedora, Red Hat)

On a local host running Red Hat 8.x, run:

```
# sudo dnf install python3-netaddr
```

On a local host running Red Hat 7.x, run:

```
# sudo yum install python2-netaddr
```

On a local host running Fedora, run:

```
# sudo dnf install python3-netaddr
```

Re-run the Ansible playbook

```
$ ansible-playbook -i inventory/hosts playbook.yml
```

6.5. Shared connection closed on provision host when installing packages

When deploying in an environment where subscription manager is not being used and a local repository is being setup on the provision host due to the nature that the provision host is offline, you may see the following error.

```
TASK [node-prep : Install required packages]
**********************************************************************************
**********
Thursday 07 May 2020  17:04:21 +0000 (0:00:00.152)       0:00:11.854 **********
fatal: [provisioner.example.com]: FAILED! => {"changed": false, "module_stderr":
"Shared connection to provisioner.example.com closed.\r\n", "module_stdout": "[Errno
101] Network is unreachable\r\n\r\n{\"msg\": \"Nothing to do\", \"changed\": false,
\"results\": [], \"rc\": 0, \"invocation\": {\"module_args\": {\"name\": [\"
firewalld\", \"tar\", \"libvirt\", \"qemu-kvm\", \"python3-devel\", \"jq\", \"
ipmitool\", \"python3-libvirt\", \"python3-lxml\", \"python3-yaml\", \"NetworkManager-
libnm\", \"nm-connection-editor\", \"libsemanage-python3\", \"policycoreutils-
python3\", \"podman\"], \"state\": \"present\", \"update_cache\": true,
\"allow_downgrade\": false, \"autoremove\": false, \"bugfix\": false,
\"disable_gpg_check\": false, \"disable_plugin\": [], \"disablerepo\": [],
\"download_only\": false, \"enable_plugin\": [], \"enablerepo\": [], \"exclude\": [],
\"installroot\": \"/\", \"install_repoquery\": true, \"install_weak_deps\": true,
\"security\": false, \"skip_broken\": false, \"update_only\": false, \"
validate_certs\": true, \"lock_timeout\": 30, \"conf_file\": null, \"
disable_excludes\": null, \"download_dir\": null, \"list\": null, \"releasever\":
null}}}\r\n", "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error", "rc": 0}
```

The error basically means that `dnf` was not able to load particular plugins, specifically the `product-id` and the `subscription-manager` plugins. However,since this is a local repository with offline access, we will want to disable these plugins when this error occurs.

On the provision host, if you run the following command:

```
[kni@provisioner ~]$ sudo dnf info dnf
Updating Subscription Management repositories.
Unable to read consumer identity
[Errno 101] Network is unreachable
Last metadata expiration check: 0:08:49 ago on Thu 07 May 2020 08:11:19 PM UTC.
Installed Packages
Name         : dnf
Version      : 4.2.7
Release      : 7.el8_1
Architecture : noarch
Size         : 1.7 M
Source       : dnf-4.2.7-7.el8_1.src.rpm
Repository   : @System
From repo    : rhel-8-for-x86_64-baseos-rpms
Summary      : Package manager
URL          : https://github.com/rpm-software-management/dnf
License      : GPLv2+ and GPLv2 and GPL
Description  : Utility that allows users to manage packages on their systems.
             : It supports RPMs, modules and comps groups & environments.
```

To ensure the issue is plugin related, we can attempt to run the same command with plugins

disabled as such:

```
[kni@provisioner ~]$ sudo dnf info dnf --disableplugin=product-id,subscription-manager
Last metadata expiration check: 0:11:17 ago on Thu 07 May 2020 08:11:19 PM UTC.
Installed Packages
Name         : dnf
Version      : 4.2.7
Release      : 7.el8_1
Architecture : noarch
Size         : 1.7 M
Source       : dnf-4.2.7-7.el8_1.src.rpm
Repository   : @System
From repo    : rhel-8-for-x86_64-baseos-rpms
Summary      : Package manager
URL          : https://github.com/rpm-software-management/dnf
License      : GPLv2+ and GPLv2 and GPL
Description  : Utility that allows users to manage packages on their systems.
             : It supports RPMs, modules and comps groups & environments.
```

If you notice, the portion that says

```
Unable to read consumer identity
[Errno 101] Network is unreachable
```

is no longer stated.

For this fix to be permanent, modify the `/etc/yum.conf` file and include the `plugins=0` into the `[main]` section of the configuration file.

```
[kni@provisioner ~]$ cat /etc/yum.conf

[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
plugins=0
```

Chapter 7. Gotchas

7.1. Using `become: yes` within `ansible.cfg` or inside `playbook.yml`

This Ansible playbook takes advantage of the `ansible_user_dir` variable. As such, it is important to note that if within your `ansible.cfg` or within the `playbook.yml` file the privilege escalation of `become: yes` is used, this will modify the home directory to that of the root user (i.e. `/root`) instead of using the home directory of your privileged user, `kni` with a home directory of `/home/kni`

7.2. Failed to install `python3-crypto` & `python3-pyghmi`

The Ansible playbook uses the `ipmi_power` module to power off the OpenShift cluster nodes prior to deployment. This particular module has a dependency for two packages: `python3-crypto` and `python3-pyghmi`. When using Red Hat Enterprise Linux 8, these packages do not reside in BaseOS nor AppStream repositories. If using `subscription-manager`, they reside in the OpenStack repositories such as `openstack-16-for-rhel-8-x86_64-rpms`, however, to simplify the installation of these packages, the playbook uses the available versions from `trunk.rdoproject.org`.

The playbook assumes that the provision host can access `trunk.rdoproject.org` or that the rpm packages are manually installed on provision host.

When the provision host cannot reach `trunk.rdopoject.org` and the packages are not already installed on the system, the following error can be expected

```
TASK [node-prep : Install required packages]
*******************************************************************************
**********
Thursday 07 May 2020  19:11:35 +0000 (0:00:00.161)       0:00:11.940 **********
fatal: [provisioner.example.com]: FAILED! => {"changed": false, "failures": ["No
package python3-crypto available.", "No package python3-pyghmi available."], "msg":
"Failed to install some of the specified packages", "rc": 1, "results": []}
```

The `python3-crypto` and `python3-pyghmi` can be downloaded from the following links if required for an offline provision host:

- python3-crypto
- python3-pyghmi

7.3. Failed to connect to bus: No such file or directory

The Ansible playbook creates two containers (when enabled) to store a mirored registry and a caching webserver. When these containers are created, the playbook also creates a `systemd` unit file to ensure these containers are restarted upon the reboot of the host serving them.

Since these are `systemd` user services, when logging into a system to attempt a command such as `systemctl --user status container-cache.service` for the webserver or `systemctl --user status container-registry.service` for the mirrored registry, you may get an error such as:

```
[kni@provisioner ~]$ systemctl --user status container-cache
Failed to connect to bus: No such file or directory
```

What the following error is trying to address is that the parameter, `DBUS_SESSIONBUS_ADDRESS`, is not set.

In order to set this variable, we can `export` as follows:

```
export DBUS_SESSIONBUS_ADDRESS="unix:path/run/user/$id/bus"
```

Once that has been set, if you re-attempt the `systemctl` command, you should see output as follows:

```
[kni@provisioner ~]$ systemctl --user status container-cache.service
● container-cache.service - Podman container-cache.service
   Loaded: loaded (/home/kni/.config/systemd/user/container-cache.service; enabled;
vendor preset: enabled)
   Active: active (running) since Mon 2020-06-01 19:52:04 UTC; 49min ago
  Process: 36380 ExecStart=/usr/bin/podman start rhcos_image_cache (code=exited,
status=0/SUCCESS)
 Main PID: 36410 (conmon)
```

Appendix A: Using Ansible Tags with the `playbook.yml`

As this playbook continues to grow, there may be times when it is useful to run specific portions of the playbook rather than running everything the Ansible playbook offers.

For example, a user may only want to run the networking piece of the playbook or create just the pull-secret.txt file, or just clean up the environment — just to name a few.

As such the existing playbook has many tags that can be used for such purposes. By running the following command you can see what options are available.

```
$ ansible-playbook -i inventory/hosts playbook.yml --list-tasks --list-tags

playbook: playbook.yml

  play #1 (provisioner): IPI on Baremetal Installation Playbook TAGS: []
    tasks:
      include_tasks TAGS: [validation]
      include_tasks TAGS: [subscription]
      include_tasks TAGS: [packages]
      include_tasks TAGS: [network]
      include_tasks TAGS: [network_facts]
      include_tasks TAGS: [user]
      include_tasks TAGS: [services]
      include_tasks TAGS: [firewall]
      include_tasks TAGS: [storagepool]
      include_tasks TAGS: [clusterconfigs]
      include_tasks TAGS: [powerservers]
      include_tasks TAGS: [cleanup, getoc]
      include_tasks TAGS: [extract, pullsecret]
      include_tasks TAGS: [rhcospath]
      include_tasks TAGS: [cache]
      include_tasks TAGS: [installconfig]
      include_tasks TAGS: [metal3config]
      include_tasks TAGS: [customfs]
      include_tasks TAGS: [manifests]
      include_tasks TAGS: [extramanifests]
      include_tasks TAGS: [cleanup]
      include_tasks TAGS: [install]
      TASK TAGS: [cache, cleanup, clusterconfigs, customfs, extract, extramanifests,
firewall, getoc, install, installconfig, manifests, metal3config, network,
network_facts, packages, powerservers, pullsecret, rhcospath, services, storagepool,
subscription, user, validation]
```

To break this down further, the following is a description of each tag.

Table 1. Table Playbook Tag Description

tag	description
`validation`	It is ***always*** required. It verifies that everything in your environment is set and ready for OpenShift deployment and sets some required internal variables
`subscription`	subscribe via Red Hat subscription manager
`packages`	install required package for OpenShift
`network`	setup the provisioning and baremetal network bridges and bridge slaves
`network_facts`	regather networking facts of environment
`user`	add remote user to `libvirt` group and generate SSH keys
`services`	enable appropriate services for OpenShift
`firewall`	set firewall rules for OpenShift
`storagepool`	define, create, auto start the default storage pool
`clusterconfigs`	directory that stores all configuration files for OpenShift
`powerservers`	power off all servers that will be part of the OpenShift cluster
`getoc`	get the appropriate `oc` binary, extract it and place within `/usr/local/bin`
`extract`	extract the OpenShift installer
`pullsecret`	copy the `pullsecret` to the `pull-secret.txt` file under the remote user home directory
`rhcospath`	set the RHCOS path
`cache`	tasks related to enabling RHCOS image caching
`installconfig`	generates the install-config.YAML
`metal3config`	generates the metal3-config.YAML
`customfs`	deals with customizing the filesystem via ignition files
`manifests`	create the manifests directory
`extramanifests`	include any extra manifests files
`install`	Deploy OpenShift
`cleanup`	clean up the environment within the provisioning node. Does not remove networking

A.1. How to use the Ansible tags

The following is an example on how to use the `--tags` option. In this example, we will just install the packages to the provision node.

Example 1

```
ansible-playbook -i inventory/hosts playbook.yml --tags "validation,packages"
```

The example above calls for the setup of the networking and installation of the packages from the Ansible playbook. Only the tasks with these specific tags will run.

Due to the dependencies in the playbook, the `validation` tag is always required.

A.2. Skipping particular tasks using Ansible tags

In the event that you want to always skip certain tasks of the playbook this can be done via the `--skip-tag` option.

We will use similar example as above where we want to skip the network setup and the package installation.

Example 1

```
ansible-playbook -i inventory/hosts playbook.yml --skip-tags "network,packages"
```

Appendix B: Using a proxy with your Ansible playbook

When running behind a proxy, it is important to properly set the environment to handle such scenario such that you can run the Ansible playbook. In order to use a proxy for the ansible playbook set the appropriate variables within your `inventory/hosts` file. These values will also be included within your generated `install-config.yaml` file.

```
# (Optional) Provide HTTP proxy settings
#http_proxy=http://USERNAME:PASSWORD@proxy.example.com:8080

# (Optional) Provide HTTPS proxy settings
#https_proxy=https://USERNAME:PASSWORD@proxy.example.com:8080

# (Optional) comma-separated list of hosts, IP Addresses, or IP ranges in CIDR format
# excluded from proxying
# NOTE: OpenShift does not accept '*' as a wildcard attached to a domain suffix
# i.e. *.example.com
# Use '.' as the wildcard for a domain suffix as shown in the example below.
# i.e. .example.com
#no_proxy_list="172.22.0.0/24,.example.com"
```