# OpenStack-Ansible Documentation: os_ironic role

***Release 18.1.0.dev304***

**OpenStack-Ansible Contributors**

**Feb 16, 2024**

# CONTENTS

CHAPTER

ONE

# CONFIGURING THE BARE METAL (IRONIC) SERVICE (OPTIONAL)

This section describes the general concepts involved in an Ironic deployment. It will be necessary for a deployer to make decisions about how to apply the general concepts in their own environment to meet a specific set of requirements. It should be understood that the Ironic service is highly configurable and pluggable so there is no single reference design specified by the Ironic project team.

**Note:** This feature is experimental at this time and has not been fully production tested.

Ironic is an OpenStack project which provisions bare metal (as opposed to virtual) machines by leveraging common technologies such as PXE boot and IPMI to cover a wide range of hardware, while supporting pluggable drivers to allow vendor-specific functionality to be added.

OpenStacks Ironic project makes physical servers as easy to provision as virtual machines in a cloud.

## 1.1 OpenStack-Ansible Deployment

The use of Ironic within an OpenStack deployment leverages Nova to deploy baremetal instances when an `openstack server create` command is issued using a baremetal flavor. So, in addition to Ironic API services, a Nova compute service using an Ironic compute driver (as opposed to libvirt) must be configured. The playbooks can automatically deploy this service when the inventory is configured accordingly.

To deploy Ironic, populate the respective groups within `openstack_user_config.yml`:

```
ironic-infra_hosts: *infrastructure_hosts
ironic-compute_hosts: *infrastructure_hosts
```

With the inventory updated, Ironic API and conductor services will be deployed on the infra/controller nodes, along with a `nova-compute` service configured for use with Ironic.

OpenStack-Ansible is configured to support PXE-based deployments by default. To enable the use of iPXE, which uses HTTP instead of TFTP for the full deployment, add the following override:

```
ironic_ipxe_enabled: yes
```

**Note:** With iPXE enabled, PXE is used to bootstrap into the iPXE loader. Deployment times are considerably faster with iPXE vs PXE, and its configuration is highly recommended. When iPXE is enabled, a web server is deployed on the conductor node(s) to host images and files.

Some drivers of the Baremetal service (in particular, any drivers using Direct deploy or Ansible deploy interfaces, and some virtual media drivers) require target user images to be available over clean HTTP(S) URL with NO authentication involved (neither username/password-based, nor token-based).

The default deploy method relies on Swift to provide this functionality. If Swift is not available in your environment, then the following override can provide similar functionality by using the web server deployed the conductor node(s) (see `ironic_ipxe_enabled`):

```
ironic_enable_web_server_for_images: yes
```

The Ironic `ipmi` hardware driver is enabled by default. Vendor-specific drivers, including iLO and DRAC, are available for use with supported hardware. OpenStack-Ansible provides a set of drivers with pre-configured hardware, boot, deploy, inspect, management, and power characteristics, including:

- agent_ilo
- agent_ipmitool
- agent_ipmitool_socat
- agent_irmc
- pxe_drac
- pxe_drac_inspector
- pxe_ilo
- pxe_ipmitool
- pxe_ipmitool_socat
- pxe_irmc
- pxe_snmp

---

**Note:** The characteristics of these drivers can be seen in further details by reviewing the `ironic_driver_types` variable in the Ironic role.

---

To enable iLO and DRAC drivers, along with IPMI, set the following override:

```
ironic_drivers_enabled:
  - agent_ipmitool
  - pxe_ipmitool
  - agent_ilo
  - pxe_ilo
  - pxe_drac
```

## 1.2 Setup Neutron Networks for Use With Ironic

Ironic supports two main network interfaces: `flat` and `neutron`:

- The `flat` interface places all provisioned nodes and nodes being deployed into a single layer 2 network.
- The `neutron` interface provides tenant-defined networking (aka multi-tenancy) by integrating with Neutron, while also separating tenant networks from the provisioning and cleaning provider networks.

With the `flat` network interface, inspection, cleaning, and provisioning functions will be performed on the same `flat` provider network. All baremetal nodes will share the same VLAN and network/subnet once deployed, which may present security challenges to tenants and to the control plane.

With the `neutron` network interface, inspection, cleaning, provisioning, and tenant networks can use distict VLANs. However, an ML2 plugin such as `networking-generic-switch` must be used to configure the respective switchports when switching between functions.

https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-genericswitch.html

---

**Note:** Both the `flat` and `neutron` network interfaces require a cleaning network to be defined in `ironic.conf`. For `flat` deployments, the cleaning network will be the same as the deployment network.

---

Create a network and subnet to be used by the baremetal instance for cleaning, provisioning, and post-deployment use:

```
openstack network create \
--provider-network-type flat \
--provider-physical-network physnet1 \
myBaremetalNetwork

openstack subnet create \
--network myBaremetalNetwork \
--subnet-range 172.17.100.0/24 \
myBaremetalNetworkSubnet
```

Set an override to define the cleaning network name:

```
ironic_neutron_cleaning_network_name: "myBaremetalNetwork"
```

---

**Note:** Ironic multi-tenancy is an advanced topic that requires the use of a compatible ML2 driver such as `networking-generic-switch`.

---

---

**Important:** Provisioning activities on baremetal instances require network access to the Ironic conductor (web) service and other OpenStack APIs. You must ensure routing exists between respective networks for deployments to succeed.

---

## 1.3 Building Ironic Images

Bare Metal provisioning requires two sets of images: the deploy images and the user images. The deploy images consist of a kernel and ramdisk image that are used by Ironic to prepare the baremetal server for actual OS deployment, whereas the user images are installed on the baremetal server to be used by the end user.

For more information on building and uploading disk images for use with Ironic, refer to the following documentation:

https://docs.openstack.org/ironic/latest/user/creating-images.html https://docs.openstack.org/ironic/latest/install/configure-glance-images.html

There are two types of user images:

- Partition Images
- Whole Disk Images

For your convenience, the following steps have been provided to demonstrate creating partition-based images.

---

**Note:** Images created using `diskimage-builder` must be built outside of an LXC container. For this process, use one of the physical hosts within the environment or a virtual machine.

---

1. Install the necessary pre-requisites:

```
apt install qemu uuid-runtime curl
```

2. Install the `disk-imagebuilder` package:

```
pip install diskimage-builder
```

---

**Important:** Only use the `--isolated` flag if you are building on a node deployed by OpenStack-Ansible, otherwise pip will not resolve the external package.

---

3. Create Ubuntu Focal kernel, ramdisk, and user images:

```
export IMAGE_NAME=my-image
export DIB_RELEASE=focal
export DIB_CLOUD_INIT_DATASOURCES="Ec2, ConfigDrive, OpenStack"
disk-image-create ubuntu baremetal dhcp-all-interfaces grub2 -o ${IMAGE_
↪NAME}
```

4. Upload the created user images into the Image (Glance) Service:

```
# Kernel image:
openstack image create my-image.kernel \
--public \
--disk-format aki \
--container-format aki \
```

(continues on next page)

(continued from previous page)

```
--file my-image.vmlinuz

# Ramdisk image
openstack image create my-image.initrd \
--public \
--disk-format ari \
--container-format ari \
--file my-image.initrd

# User image
openstack image create my-image \
--public \
--disk-format qcow2 \
--container-format bare \
--property kernel_id=<kernel image uuid> \
--property ramdisk_id=<ramdisk image uuid> \
--file my-image.qcow2
```

**Note:** When a baremetal instance is provisioned using a partition-based image, the kernel and ramdisk images will be used for PXE when the `local` boot capability is not available.

## 1.4 Creating an Ironic Flavor

The use of flavors are necessary when creating instances using Nova, and baremetal flavors should be used when targeting baremetal nodes for instances. The properties of the flavor, along with the defined resource class, are useful to the scheduler when scheduling against libvirt or ironic compute services.

As an example, imagine an Ironic deployment has the following nodes:

```
- node-1:
  resource_class: ironic-gold
  properties:
    cpus: 32
    memory_mb: 32768
    capabilities:
      boot_mode: uefi,bios
- node-2:
  resource_class: ironic-silver
  properties:
    cpus: 16
    memory_mb: 16384
```

The operator might define the flavors as such:

```
- baremetal-gold
  resources:
    ironic-gold: 1
```

(continues on next page)

(continued from previous page)

```
  extra_specs:
    capabilities: boot_mode:bios
- baremetal-gold-uefi
  resources:
    ironic-gold: 1
  extra_specs:
    capabilities: boot_mode:uefi
- baremetal-silver
  resources:
    ironic-silver: 1
```

A user booting an instance with either the baremetal-gold or baremetal-gold-uefi flavor would land on node-1, because capabilities can still be passed down to ironic, and the resource_class on the node matche what is required by flavor. The baremetal-silver flavor would match node-2.

---

**Note:** A flavor can request exactly one instance of a bare metal resource class.

---

When creating a baremetal flavor, its useful to add the RAM and CPU properties as a convenience to users, although they are not used for scheduling. In addition, the DISK property is also not used for scheduling, but is still used to determine the root partition size.

```
openstack flavor create \
--ram 32768 \
--vcpu 32 \
--disk 120 \
baremetal-gold
```

After creation, associate each flavor with one custom resource class. The name of a custom resource class that corresponds to a nodes resource class (in the Bare Metal service) is:

- the bare metal nodes resource class all upper-cased
- prefixed with `CUSTOM_`
- all punctuation replaced with an underscore

```
openstack flavor set \
--property resources:CUSTOM_IRONIC_GOLD=1 \
baremetal-gold
```

---

**Note:** Ensure the resource class defined in the flavor matches that of the baremetal node, otherwise, the scheduler will not find eligible hosts. In the example provided, the resource class is `ironic-gold`.

---

Another set of flavor properties must be used to disable scheduling based on standard properties for a bare metal flavor:

```
openstack flavor set --property resources:VCPU=0 baremetal-gold
openstack flavor set --property resources:MEMORY_MB=0 baremetal-gold
openstack flavor set --property resources:DISK_GB=0 baremetal-gold
```

Lastly, a `boot_option` capability can be set to speed up booting after the deployment:

```
openstack flavor set --property capabilities:'boot_option=local' baremetal-
↪gold
```

**Note:** Specifying the `local` boot option allows the deployed baremetal instance to boot directly to disk instead of network.

## 1.5 Enrolling Ironic Nodes

Enrolling baremetal nodes makes then available to the Ironic service. The properties of a given node will allow Ironic to determine how an image should be deployed on the node, including using IPMI or vendor-specific out-of-band interfaces. Some properties are optional, and may rely on defaults set by the operator or within OpenStack-Ansible. Others are required, and may be noted as such.

Some things should be known about the baremetal node prior to enrollment, including:

- Node Name
- Driver
- Deploy Interface (based on driver)
- Provisioning Interface (MAC Address)
- IPMI or OOB Credentials
- OOB Management IP
- Deploy Kernel Image UUID (from Glance)
- Deploy Ramdisk Image UUID (from Glance)
- Boot Mode (bios or uefi)
- Network Interface (flat or neutron)

**Note:** Kernel and ramdisk images may be provided by the `diskimage-builder` process, or may be downloaded from opendev.org:

https://tarballs.opendev.org/openstack/ironic-python-agent/dib/ https://docs.openstack.org/ironic/latest/install/deploy-ramdisk.html

**Important:** The deploy kernel and ramdisk should be updated on a regular basis to match the OpenStack release of the underlying infrastructure. The Ironic Python Agent that runs on the ramdisk interfaces with Ironic APIs, and should be kept in sync.

To enroll a node, use the `openstack baremetal node create` command. The example below demonstrates the creation of a baremetal node with the following characteristics:

```
node_name=baremetal01
node_mac="f0:92:1c:0c:1f:88"    # MAC address of PXE interface (em1 as␣
↪example)
deploy_aki=ironic-deploy-aki    # Kernel image
deploy_ari=ironic-deploy-ari    # Ramdisk image
resource=ironic-gold            # Ironic resource class (matches flavor as␣
↪CUSTOM_IRONIC_GOLD)
phys_arch=x86_64
phys_cpus=32
phys_ram=32768
phys_disk=270
ipmi_username=root
ipmi_password=calvin
ipmi_address=172.19.0.22
boot_mode=bios
network_interface=flat
```

**Important:** The Ironic conductor service must be able to communicate with the OOB IP address to perform provisioning functions.

```
openstack baremetal node create \
  --driver ipmi \
  --deploy-interface direct \
  --driver-info ipmi_username=$ipmi_username \
  --driver-info ipmi_password=$ipmi_password \
  --driver-info ipmi_address=$ipmi_address \
  --driver-info deploy_kernel=`openstack image show $deploy_aki -c id |awk '/
↪id / {print $4}'` \
  --driver-info deploy_ramdisk=`openstack image show $deploy_ari -c id |awk '/
↪id / {print $4}'` \
  --property cpus=$phys_cpus \
  --property memory_mb=$phys_ram \
  --property local_gb=$phys_disk \
  --property cpu_arch=$phys_arch \
  --property capabilities='boot_option:local,disk_label:gpt' \
  --resource-class $resource \
  --network-interface $network_interface \
  --name $node_name
```

The node will first appear in an `enroll` state. To make it available for provisioning, set the state to `manage`, then `available`:

```
openstack baremetal node manage baremetal01
openstack baremetal node provide baremetal01
openstack baremetal node list --fit

+--------------------------------------+-------------+---------------+--------
↪-----+--------------------+-------------+
```

(continues on next page)

(continued from previous page)

```
| UUID                                 | Name        | Instance UUID | Power␣
↪State | Provisioning State | Maintenance |
+--------------------------------------+-------------+---------------+--------
↪-----+--------------------+-------------+
| c362890d-5d7a-4dc3-ad29-7dac0bf49344 | baremetal01 | None          | power␣
↪off   | available          | False       |
+--------------------------------------+-------------+---------------+--------
↪-----+--------------------+-------------+
```

Next, create a baremetal port using the `openstack baremetal port create` command:

```
node_name=baremetal01
node_mac="f0:92:1c:0c:1f:88"
openstack baremetal port create $node_mac \
--node `openstack baremetal node show $node_name -c uuid |awk -F "|" '/ uuid ␣
↪/ {print $3}'`

+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| address               | f0:92:1c:0c:1f:88                    |
| created_at            | 2021-12-17T20:36:19+00:00            |
| extra                 | {}                                   |
| internal_info         | {}                                   |
| is_smartnic           | False                                |
| local_link_connection | {}                                   |
| node_uuid             | c362890d-5d7a-4dc3-ad29-7dac0bf49344 |
| physical_network      | None                                 |
| portgroup_uuid        | None                                 |
| pxe_enabled           | True                                 |
| updated_at            | None                                 |
| uuid                  | 44e5d872-ffa5-45f5-a5aa-7147c523e593 |
+-----------------------+--------------------------------------+
```

**Note:** The baremetal port is used to setup Neutron to provide DHCP services during provisioning. When the `neutron` network interface is used, the respective switchport can be managed by OpenStack.

## 1.6 Deploy a Baremetal Node Using Ironic

Baremetal instances can be deployed using the `openstack server create` command and a baremetal flavor. Unless the image has been created with support for passwords, an SSH key must be provided. The baremetal instance relies on Neutron DHCP and metadata services, just like a virtual instance.

```
openstack server create \
--flavor baremetal-gold \
--image focal-server-cloudimg-amd64 \
```

(continues on next page)

(continued from previous page)

```
--key-name myKey \
--network myBaremetalNetwork \
myBaremetalInstance
```

**Important:** If you do not have an ssh key readily available, set one up with `ssh-keygen` and/or create one with `openstack keypair create`. Otherwise, you will not be able to connect to the deployed instance.

CHAPTER

TWO

# EXAMPLE LXC BASED IRONIC DEPLOYMENT

This section describes a specific deployment of Ironic using Openstack-Ansible. A number of design choices are made which illustrate how to configure the Ironic service for a specific set of requirements.

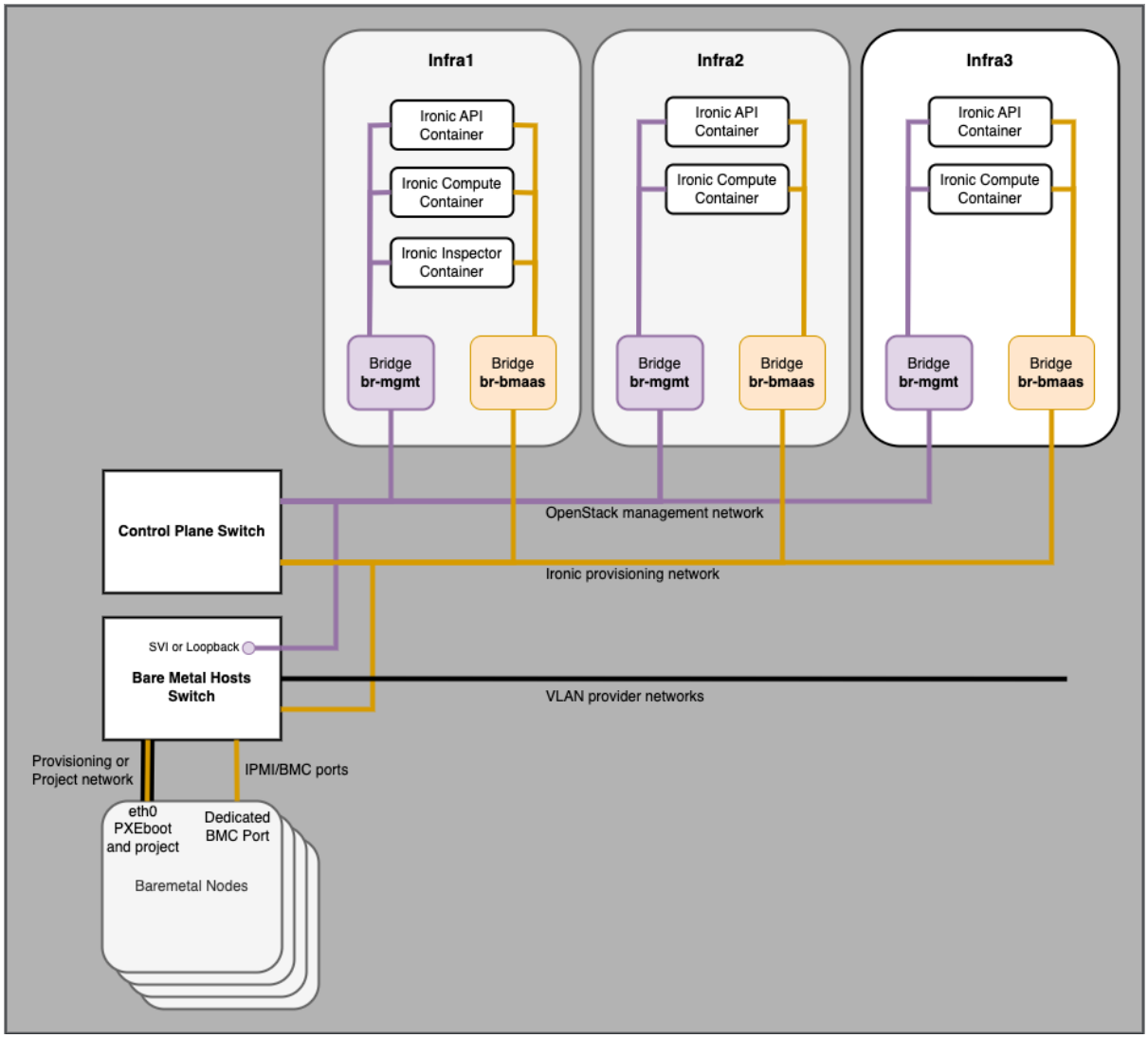
Deployment design decisions:

- LXC containers are used in the openstack control plane
- A single `bmaas` network is used for Ironic provisioning, cleaning and inspection
- The `bmaas` network is not routable to any other networks nor to the internal VIP
- Multitenancy is used with Ironic servers attached to project networks
- networking-generic-switch will be used to control network devices to achieve multitenancy
- Cisco NXOS switches
- The deployment uses VXLAN project networks

A number of these design decisions are opinionated and could be changed, e.g. allowing the `bmaas` network to be routed to other networks including the internal VIP would result in some simplification.

This example is illustrative of a specific set of deployment requirements but is not intended to be followed rigidly. It demonstrates some of the capabilities of Ironic and Openstack-Ansible and how to approach most parts of a practical deployment of Ironic at small to modest scales.

**Warning:** Consideration should be given to the security of IPMI or other out-of-band interfaces which are notoriously buggy and often have vendor specific in-band tools which allow the BMC and its firmware to be manipulated from userspace. Understand the risks of allowing IPMI/BMC interfaces to share a physical interface with onboard ethernet ports, this feature will allow full access to the management interface of a deployed Ironic node. For high security environments or where the users are untrusted consult your hardware vendor for appropriate hardening steps.




## 2.1 BMAAS network address plan

In this example the subnet for the `bmaas` network is 10.88.104.0/24. The size of the subnet determines the maximum number of nodes in the Ironic deployment.

| Address | Purpose |
|---|---|
| 10.88.104.0 to .10 | Reserve for potential physical routers / SVI |
| 10.88.104.11 to .29 | Control plane Ironic container interfaces |
| 10.88.104.64 to .95 | Neutron DHCP allocation pool in Ironic_Network |
| 10.88.104.96 to .127 | Ironic Inspector DHCP pool range |
| 10.88.104.128 to .254 | Static IPs manually assigned to IPMI/iDRAC |

In this deployment the `bmaas` network combines several functions for simplicity. It would be possible to use separate networks for inspection, cleaning and provisioning but that is beyond the scope of this example.

The subnet is divided into several address ranges, a reservation for future interfaces on physical routers, or a gateway address if the subnet is to be made routable in the future. A small number of addresses are then reserved for the `bmaas` network to connect to the Ironic containers in the control plane, and the remainder

of the addresses are shared between a neutron network for provisioning, a range for Ironic Inspector to allocate with DHCP and finally a block of addresses for the IPMI (or other protocol) management ports of the Ironic nodes.

---

**Note:** This example will use VLAN ID 3003 for the `bmaas` network but any available VLAN ID could be used.

---

**Warning:** This example collapses the Ironic IPMI, provisioning, cleaning and inspection networks into the same subnet. It is possible to make these be separate from each other by creating a network for each different function and updating the container networks and Ironic service configuration suitably. In particular it should be understood that the nodes are untrusted during the cleaning phase and will be in an arbitrary state as left by the previous user.

## 2.2 IPMI Interfaces

When configuring IPMI interfaces for Ironic nodes, the following steps are recommended:

- Use static IP allocations for the IPMI interfaces, unless there is already a very reliable means of allocating addresses with DHCP. The Ironic team do not consider the Neutron DHCP agent to be suitable for assigning addresses to the IPMI interfaces.
- Use dedicated IPMI ports on Ironic nodes especially if multitenancy is required. A node with a shared onboard ethernet/IPMI port will have that port moved into the tenant network when deployment is complete and the Ironic control plane will no longer be able to communicate with the management interface of the node.

## 2.3 Maximum size of the deployment

The maximum size of this Ironic deployment is limited by the address alloction in the `bmaas` network. In this example there can be a maximum of 127 server BMC interfaces in the range 10.88.104.128/25.

The maxiumum number of servers that can be simultanously provisioned is determined by the address allocation to the Neutron DHCP pool.

The maximum number of servers that can be simultanously inspected is determined by the address allocation to the Ironic Inspector DHCP pool.

To increase the size of the deployment, the size of the `bmaas` subnet should be increased and the addresses allocated to meet the number of required nodes and maximum number of simultaneous nodes being provisioned and inspected.

## 2.4 Openstack-Ansible configuration

Once the address plan has been decided, the Openstack-Ansible configuration can be updated to match.

The existing `cidr_networks` and `used_ips` sections in `/etc/openstack_deploy/openstack_user_config.yml` must have extra entries to describe the network range available for the Ansible inventory to assign to Ironic control plane containers, in this example all addresses in the `bmaas` network are marked as used except the range 10.88.104.11 to 10.88.104.29.

An additional network is defined in the `provider_networks` list which represents the connection between the bridge `br-bmaas` on the controller and `eth15` inside the ironic service containers.

The `bmaas` network must be extended from the control plane hosts to the switch ports connected to the Ironic node IPMI interfaces, and also to switch ports connected to the interfaces on the Ironic nodes that will be used for PXEboot. This will typically be a VLAN allocated specifically for Ironic provisioning.

The hosts for the Ironic control plane containers are assigned.

---

**Note:** It is the responsibility of the deployer to create br-bmaas on the controller nodes and ensure that it is connected to the correct VLAN ID for the `bmaas` network. Configuration of host networking is outside the scope of Openstack-Ansible.

---

---

**Note:** The `range` key in the provider network definition is not used but its useful as an reminder in the config file of the VLAN ID.

---

```
cidr_networks:
  <existing entries>
  bmaas:     10.88.104.0/24        # for containers on the bmaas network

used_ips:
  <existing entries>
  # bmaas ips
  - "10.88.104.0,10.88.104.10"     # reserve for routers or other␣
↪infrastructure
  - "10.88.104.30,10.88.104.255"   # reserve for ironic IPMI and provisioning

provider_networks:
  <existing entries>
  # Network definition to connect Ironic LXC containers to the bmaas network
  # on the infra hosts
  - network:
      net_name: physnet_neutron
      container_type: "veth"
      container_bridge: "br-bmaas"
      container_interface: "eth15"
      ip_from_q: bmaas
      type: "vlan"
      range: "3003:3003"
      group_binds:
```

(continues on next page)

(continued from previous page)

```
        - ironic_api_container
        - ironic_compute_container
        - ironic_inspector_container


# ironic API and conductor
ironic-infra_hosts:
  infra1: *_infra1_
  infra2: *_infra2_
  infra3: *_infra3_

# nova used by ironic for machine state management
ironic-compute_hosts:
  infra1: *_infra1_
  infra2: *_infra2_
  infra3: *_infra3_

# Ironic-inspector can only support a single instance at the moment
# High availability for ironic-inspector is not yet implemented
ironic-inspector_hosts:
  infra1: *_infra1_
```

**Note:** This example uses YAML Anchors to simplify `openstack_user_config.yml` allowing the IP addresses of the infra nodes to be defined only once. See https://yaml.org/spec/1.2.2/#alias-nodes.

## 2.5 Create the Neutron configuration

Enable the Neutron `baremetal` and `genericswitch` mechanism drivers by updating `/etc/openstack_deploy/group_vars/neutron_server.yml`

```
---
neutron_plugin_types:
  - ml2.genericswitch
  - ml2.baremetal

# keep the ml2 drivers in this order
# see https://storyboard.openstack.org/#!/story/2008686
neutron_ml2_mechanism_drivers: "genericswitch,baremetal"
```

Configure neutron networking-generic-switch to know about the switches that the Ironic nodes are connected to in `/etc/openstack_deploy/user_variables.yml`. These switches are programmed by neutron to switch the Ironic nodes between the provisioning and project networks once deployment is complete. This is enabling multitenancy for Ironic.

This example is for a Cisco NXOS based switch, which uses the same command set as a Cisco IOS based switch for the functions needed by networking-generic-switch. There is no specific `device_type` for NXOS.

**Note:** A MAC address for the switch must be specified in the neutron config, but Cisco and some other switch vendors present a unique MAC address per port so the MAC address as seen from the client cannot be used to identify the switch. For IOS/NXOS networking-generic-switch uses the field `switch_info` from the Ironic node `local_link_connection` information rather than match a MAC address when choosing which switch to configure for a particular node.

```
neutron_neutron_conf_overrides:
  genericswitch:my-switch-name:            # This should match the hostname␣
↪configured on the switch
    device_type: netmiko_cisco_ios         # It is really NXOS but the␣
↪commands are the same
    ngs_mac_address: "cc:46:d6:64:4b:41"   # Doesn't seem to matter but is␣
↪required - this is taken from an SVI on the mgmt network
    ip: "10.80.240.3"                      # An IP on the switch which has␣
↪ssh access from the br-mgmt network, loopback, SVI or mgmt0 as needed
    username: "neutron"                    # The user that Neutron will SSH␣
↪to the switch as
    password: "supersecret"                # The password that Neutron will␣
↪use to SSH to the switch
    ##  key_file: <ssh key file>           # An SSH key may be used instead␣
↪of a password
    ngs_manage_vlans: "False"              # VLANs are already provisioned on␣
↪the switch so tell neutron not to create/delete VLANs
```

**Note:** The configuration for networking-generic-switch is added to neturon.conf rather than ml2_conf_genericswitch.ini as the config needs to be read by both neutron-rpc-server and neutron-server. neutron-server is a uwsgi service in openstack-ansible so is only passed one config file, see https://bugs.launchpad.net/openstack-ansible/+bug/1987405

**Note:** If there is already an override in place for this variable then extend the existing override rather than making a second one.

## 2.6 Configure switch to allow ssh from Neutron

To achieve multitenancy, Neutron will connect to the specified switch and configure the port for the Ironic node being provisioned to be in the correct project VLAN once the deployment is complete. During deployment Neutron will ensure that the node is in the `bmaas` provisioning network as specified in the Ironic config.

A suitable user and credential must exist on the switch. The SSH connection will originate from the Neutron processes running on the OpenStack control plane, on the `mgmt` network. There must be an IP route from the `mgmt` network to an interface on the switch which permits SSH login. That interface could be a physical management port (`mgmt0` on NXOS), a loopback interface, an SVI or another interface with an IP address. SSH communication with the switch can happen either in-band or out-of-band depending on the requirements of the deployment.

This example config is for a `neutron` user using password authentication on an NXOS switch as seen by `show run`. The config applied on the switch gives the `neutron` user access to a minimal set of commands for configuring VLAN membership on specific ports.

To control the commands that the `neutron` user is allowed to issue on the Cisco Nexus switch create a role:

```
role name neutron-role
  rule 3 permit command configure t
  rule 2 permit read-write feature interface
  rule 1 permit read
  vlan policy deny
    permit vlan 3003-3003
    permit vlan 3100-3200
  interface policy deny
    permit interface Ethernet1/1
    permit interface Ethernet1/2
    permit interface Ethernet1/3
    permit interface Ethernet1/4
    permit interface Ethernet1/5
    permit interface Ethernet1/6
    permit interface Ethernet1/7
    permit interface Ethernet1/8
```

This role allows the `neutron` user assign a port to VLAN 3003 which is the `bmaas` network and is used during node provisioning. Any project VLANS that nodes should be able to be moved into after deployment should also be permitted, range 3100-3200 here.

The interfaces which the `neutron` user is permitted to modify are listed, in this case individually but consult the switch documentation for other options such as a regular expression.

A similar config can be made on an Arista switch, where a much more explicit list of allowed CLI commands must be defined using regular expressions.

```
role neutron-role
   10 permit mode exec command configure
   20 permit mode exec command terminal width 511
   30 permit mode exec command terminal length 0
   40 permit mode exec command enable
   50 permit mode exec command copy running-config startup-config
   60 permit mode config command interface
   70 permit mode if-Et([1-9]|27|29)\/1 command switchport mode access
   80 permit mode if-Et([1-9]|27|29)\/1 command (no )*switchport access vlan␣
↪(3003|3966)
   90 permit mode if-Et([1-9]|27|29)\/1 command no switchport mode trunk
   100 permit mode if-Et([1-9]|27|29)\/1 command switchport trunk allowed␣
↪vlan none
   110 permit mode config command copy running-config startup-config
```

Create the user and password, which must match those in the `neutron.conf / genericswitch` config file options:

```
username neutron password 5 <ENCRYPTED-PASSWORD-HERE> role neutron-role
```

Allow SSH to the switch from the expected IP addresses, for example a pair out of band management hosts 192.168.0.100/31 and the OpenStack `mgmt` network 10.80.240.0/24.

```
ip access-list ACL_ALLOW_SSH_VTY
  10 permit tcp 192.168.0.100/31 any eq 22
  20 permit tcp 10.80.240.0/22 any eq 22

line vty
  session-limit 5
  exec-timeout 10
  access-class ACL_ALLOW_SSH_VTY in
```

## 2.7 Create the Neutron network for Ironic provisioning, cleaning and inspection

```
openstack network create \
  --internal \
  --provider-network-type vlan \
  --provider-physical-network physnet_neutron \
  --provider-segment 3003 \
  Ironic_Network

openstack subnet create \
  --allocation-pool 10.88.104.64-10.88.104.95 \
  --dhcp \
  --subnet-range 10.88.104.0/24
  --gateway none
  Ironic_Subnet
```

## 2.8 Create the Ironic configuration

In `/etc/openstack_deploy/user_variables_ironic.yml`

```
## IRONIC ##

ironic_ipxe_enabled: yes                    # use HTTP image download from the␣
↪ironic conductor container
ironic_enable_web_server_for_images: yes  # use same web server to cache user␣
↪images

# Ensure values used during PXEboot refer directly to the correct interface␣
↪on Ironic API container
# instead of the internal VIP
ironic_http_url: "{{ ironic_ipxe_proto }}://{{ container_networks['bmaas_
↪address']['address'] }}:{{ ironic_ipxe_port }}"
```

(continues on next page)

(continued from previous page)

```
ironic_tftp_server_address: "{{ container_networks['bmaas_address']['address
↪'] }}"

# Enable ironic drivers
ironic_drivers_enabled:      # Use PXE boot and IPMItool
  - agent_ipmitool
  - pxe_ipmitool
  - pxe_drac                 # enables drivers for Dell iDrac interface

# Configure Ironic to use Neutron networking
ironic_enabled_network_interfaces_list: "noop,neutron"
ironic_default_network_interface: neutron

# Enable the default set of cleaning steps
ironic_automated_clean: yes

# Configure the neutron networks that Ironc should use
ironic_neutron_provisioning_network_name: "Ironic_Network"
ironic_neutron_cleaning_network_name:     "Ironic_Network"
ironic_neutron_inspection_network_name:   "Ironic_Network"

# Ensure ironic API (using uwsgi) listens on br-bmaas for agent callbacks
# as well as the mgmt interface for the loadbalancer
ironic_uwsgi_bind_address: 0.0.0.0

# INI file overrides
ironic_ironic_conf_overrides:
  # Disable full device erasure (slow) and just metadata erasure, and replace␣
↪with "Express erasure"
  # which tries to use firmware secure-erase command, but if that fails,␣
↪reverts to metadata erasure.
  # See: https://docs.openstack.org/ironic/yoga/admin/cleaning.html#storage-
↪cleaning-options
  deploy:
    erase_devices_priority: 0
    erase_devices_metadata_priority: 0
  conductor:
    clean_step_priority_override: "deploy.erase_devices_express:5"

  # Direct IPA to callback directly to deploying ironic container (via BMAAS␣
↪network)
  # instead of going via HAProxy on mgmt network. Only applies when bmaas␣
↪network is isolated.
  service_catalog:
    endpoint_override: "http://{{ container_networks['bmaas_address']['address
↪'] }}:6385"

  # Enable ipmitool's Serial-over-LAN terminal console for baremetal nodes
  DEFAULT:
```

(continues on next page)

(continued from previous page)

```
    enabled_console_interfaces: "ipmitool-socat,no-console"

## IRONIC INSPECTOR ##

# Direct Inspector to callback directly to deploying ironic container (via␣
↪BMAAS network)
# instead of going via HAProxy on mgmt network. Only applies when bmaas␣
↪network is isolated.
ironic_inspector_callback_url: "{{ ironic_inspector_service_internaluri_proto␣
↪}}://{{ container_networks['bmaas_address']['address'] }}:{{ ironic_
↪inspector_service_port }}/v1/continue"

# Ensure inspector API (using uwsgi) listens on br-bmaas for agent callbacks
# as well as the mgmt interface for the loadbalancer
ironic_inspector_service_address: "0.0.0.0"

# dnsmasq/dhcp information for inspector
ironic_inspector_dhcp_pool_range: 10.88.104.96 10.88.104.127
ironic_inspector_dhcp_subnet: 10.88.104.0/24
ironic_inspector_dhcp_subnet_mask: 255.255.255.0
ironic_inspector_dhcp_enable_gateway: False
ironic_inspector_dhcp_enable_nameservers: False

ironic_inspector_dhcp_interface: eth15   # connected to br-bmaas on the host
```

## 2.9 Deploy Neutron changes

```
openstack-ansible playbooks/os-neutron-install.yml
```

## 2.10 Deploy the ironic-specific nova services

This deploys nova compute and nova console services to the ironic compute containers.

```
playbooks/os-nova-install.yml --limit ironic_all
```

## 2.11 Deploy changes to HAProxy

This will bring up the required Ironic, Inspector, and console endpoints.

```
openstack-ansible playbooks/haproxy-install.yml --tags haproxy_server-config
```

## 2.12 Deploy the Ironic and Inspector services

```
openstack-ansible playbooks/os-ironic-install.yml
```

## 2.13 Deploy the Horizon dashbaords for Ironic

```
openstack-ansible playbooks/os-horizon-install.yml
```

## 2.14 Using Ironic

Please refer to the general instructions in the Configuring Ironic section of this documentation.

## 2.15 VXLAN project networks

In this example Ironic multitenancy is implemented using VLANs. In an OpenStack deployment where project networks are implemented using an overlay such as VXLAN, it will not be possible to attach Ironic nodes directly to these networks. In addition, it is not possible for an end user to request that the underlying implementation is VLAN when creating a project network.

In a cloud using overlay project networks it will be necessary for the cloud administrator to create VLAN provider networks for users to attach Ironic nodes to and to share these into individual projects using Neutron RBAC.

CHAPTER

THREE

# CONFIGURING THE BARE METAL (IRONIC) INSPECTOR SERVICE (OPTIONAL)

**Note:** This feature is experimental at this time and it has not been fully production tested yet.

Ironic Inspector is an Ironic service that deploys a tiny image called ironic-python-agent that gathers information about a Bare Metal node. The data is then stored in the database for further use later. The node is then updated with properties based in the introspection data.

The inspector configuration requires some pre-deployment steps to allow the Ironic playbook to make the inspector functioning.

## 3.1 Networking

Ironic networking must be configured as normally done. The inspector and Ironic will both share the TFTP server.

Networking will depend heavily on your environment. For example, the DHCP for both Ironic and inspector will come from the same subnet and will be a subset of the typical ironic allocated range.

## 3.2 Required Overrides

```
# dnsmasq/dhcp information for inspector
ironic_inspector_dhcp_pool_range: <START> <END> (subset of ironic␣
↪IPs)
ironic_inspector_dhcp_subnet: <IRONIC SUBNET CIDR>
ironic_inspector_dhcp_subnet_mask: 255.255.252.0
ironic_inspector_dhcp_gateway: <IRONIC GATEWAY>
ironic_inspector_dhcp_nameservers: 8.8.8.8
```

## 3.3 To enable LLDP discovery of switch ports during inspection

During inspection Ironic Inspector can automatically populate information into the node `local_link_connection` which can automatically create a baremetal port for the node.

This example is suitable for switches that have a common MAC address per switch port and are identified to networking-generic-switch using the `ngs_mac_address` parameter which matches against the `switch_id` field in the Ironic node `local_link_connection` information.

Set the following variables in `/etc/openstack_deploy/user_variables.yml`

```
# enable LLDP discovery for inspector
ironic_inspector_processing_hooks: "$default_processing_hooks,lldp_basic,
↪local_link_connection"
ironic_inspector_extra_callback_parameters: "ipa-collect-lldp=1"
```

## 3.4 To enable LLDP discovery of switch system name during inspection

This example is suitable for switches that have a different MAC address per switch port and are identified to networking-generic-switch using the switch hostname which is matched against the `switch_info` field in the Ironic node `local_link_connection` information.

An additional out-of-tree Ironic Inspector plugin is needed to obtain the system name of the switch and write it to `switch_info` during inspection.

Set the following variables in `/etc/openstack_deploy/user_variables.yml`

```
# enable LLDP discovery for inspector
ironic_inspector_processing_hooks: "$default_processing_hooks,lldp_basic,
↪local_link_connection,system_name_llc"
ironic_inspector_extra_callback_parameters: "ipa-collect-lldp=1"

# stackhpc inspector plugins
ironic_inspector_user_pip_packages:
  - git+https://github.com/stackhpc/stackhpc-inspector-plugins@master
↪#egg=stackhpc-inspector-
```

CHAPTER

FOUR

# DEPLOYING MULTIPLE IRONIC NODES WITH DIFFERENT CPU ARCHITECURES

Ironic can deploy nodes with CPU architectures which do not match the CPU architecture of the control plane. The default settings for Openstack-Ansible assume an x86-64 control plane deploying x86-64 Ironic nodes.

This documentation describes how to deploy aarch64 Ironic nodes using an x86-64 control plane. Other combinations of architecture could be supported using the same approach with different variable definitions.

This example assumes that Glance is used for Ironic image storage and the Ironic control plane web server serves these for deployment.

## 4.1 Building ironic-python-agent deploy image for aarch64

There must be an ironic-python-agent kernel and initramfs built and uploaded to Glance for each architecture that needs to be deployed.

To build an aarch64 ironic-python-agent image using a Rocky Linux aarch64 host:

```
dnf install python3-virtualenv git qemu-img

virtualenv venv
source ./venv/bin/activate
pip install ironic-python-agent-builder

export DIB_REPOREF_ironic_python_agent=origin/master
export DIB_REPOREF_requirements=origin/master
ironic-python-agent-builder -o my-ipa --extra-args=--no-tmpfs --release 9-
↪stream centos
```

- Replace `origin/master` with another branch reference in order to build specific versions of IPA, for example `stable/zed`

## 4.2 Configuring Ironic for multiple architectures

This configuration assumes the use of iPXE. The settings required to support an additional architecture are minimal. Ironic has a default setting for which EFI firmware file to use that can be overridden on a per-architecture basis with the `ipxe_bootfile_name_by_arch` config setting.

On the control plane the aarch64 EFI iPXE firmware must be present in the tftp server root directory. Note that not all distributions supply packages for EFI firmware for architectures different to the host so it may be necessary to download architecture specific firmware directly from https://boot.ipxe.org

This example shows how to specify the iPXE boot firmware to use for aarch64 nodes, and where that firmware should be obtained from to populate the tftp server.

```
ironic_ironic_conf_overrides:
  # Point to aarch64 uefi firmware on aarch64 platforms
  pxe:
    ipxe_bootfile_name_by_arch: 'aarch64:ipxe_aa64.efi'

ironic_tftp_extra_content:
  - url: http://boot.ipxe.org/arm64-efi/ipxe.efi
    name: ipxe_aa64.efi
```

**Note:** You must combine this override with any existing definition of `ironic_ironic_conf_overrides`.

## 4.3 Enrolling an aarch64 node

When enrolling an aarch64 node the `boot_mode` must be uefi even if existing Ironic nodes use legacy bios boot.

An example of the node capabilities including uefi boot would be:

```
capabilities='boot_option:local,disk_label:gpt,boot_mode:uefi'
```

Enrolling an aarch64 node is exactly the same as enrolling an x86_64 node, except that `deploy_kernel` and `deploy_ramdisk` must be set to the aarch64 version of the deploy image.

## 4.4 Building an aarch64 user image

Example of building a whole-disk aarch64 user image on an existing aarch64 Ubuntu host:

```
sudo apt update
sudo apt install python3-venv qemu-utils
python3 -m venv venv
source ./venv/bin/activate
pip install diskimage-builder
DIB_RELEASE=jammy DIB_CLOUD_INIT_DATASOURCES=Ec2 disk-image-create -a arm64␣
↪ubuntu vm block-device-efi cloud-init-datasources -o baremetal-ubuntu-22.04-
↪efi-arm64.qcow2
```

(continues on next page)

(continued from previous page)

- The DIB_RELEASE=<name> environment variable tells the ubuntu element which version of Ubuntu to create an image for. This defaults to Focal if left unspecified.
- The DIB_CLOUD_INIT_DATASOURCES=Ec2 environment variable is used by the `cloud-init-datasources` element to force `cloud-init` to use its Ec2 datasource. The native OpenStack datasource cant be used because it doesnt currently have working support for bare metal instances until `cloud-init` version 23.1. (Since the OpenStack metadata service also provides an EC2 compatible API, the Ec2 datasource is a reasonable workaround. (NB: This is actually the default behaviour for Ubuntu cloud images, but for entirely unrelated reasons hence it being worth making explicit here.)

Use a similar approach on a Rocky Linux aarch64 system to build a whole-disk user image of the latest version of Rocky Linux:

```
DIB_RELEASE=9 DIB_CLOUD_INIT_DATASOURCES=Ec2 DIB_CLOUD_INIT_GROWPART_DEVICES=
↪'["/"]' disk-image-create -a arm64 rocky-container vm block-device-efi␣
↪cloud-init openssh-server cloud-init-datasources cloud-init-growpart -o␣
↪baremetal-rocky-9-efi-arm64.qcow2
```

- The DIB_RELEASE=<number> environment variable tells the rocky-container element which version of Rocky to create an image for.
- The `cloud-init` and `openssh-server` elements are essential since the Rocky container image does not include these packages. (As an aside: the `diskimage-builder` documentation erroneously claims that the `cloud-init` element only works on Gentoo, but this is not the case).
- As with Ubuntu, setting DIB_CLOUD_INIT_DATASOURCES=Ec2 and using the `cloud-init-datasources` element is necessary since the OpenStack `cloud-init` datasource doesnt work. Unlike the Ubuntu case, using the Ec2 datasource is not the default and so adding these options is essential to obtain a working image.
- DIB_CLOUD_INIT_GROWPART_DEVICES variable tells cloud-init-growpart to configure cloud-init to grow the root partition on first boot which is must be explicitly set on some OS/architecture combinations.

CHAPTER

FIVE

# DEBUGGING THE BARE METAL (IRONIC) INSPECTOR SERVICE

## 5.1 Ironic Python Agent debug logs

If IPA fails, a log file will be written to `/var/log/ironic` on the conductor node responsible for the Ironic node being deployed.

A lot of useful information is collected including a copy of the journal log from the host.

## 5.2 Pausing during a deployment

To pause the deployment, possibly to log into an Ironic node running the IPA to do diagnostic work, the node can have `maintainance` switched on either through the OpenStack CLI or Horizon web interface.

The ironic state machine will not change state again until maintainance mode is switched off, triggered by a heartbeat from IPA. Note that you will have to wait for up to one heartbeat period for the deployment to resume after switching off maintainance mode.

## 5.3 Logging into IPA

To perform diagnostic steps on an ironic node during deployment or cleaning it might be helpful to be able to log into the node from the controller after the deploy image has booted.

To build a version of the deploy image which has the `dynamic-login` element enabled, in this case building on a Rocky Linux host:

```
dnf install python3-virtualenv git qemu-img

virtualenv venv
source ./venv/bin/activate
pip install ironic-python-agent-builder

export DIB_REPOREF_ironic_python_agent=origin/stable/zed
export DIB_REPOREF_requirements=origin/stable/zed
ironic-python-agent-builder -e dynamic-login -o my-login-ipa --extra-args=--
↪no-tmpfs --release 8-stream centos
```

Once the IPA kernel and initramfs are built, upload them to glance and set them as the deploy kernel/initramfs for the Ironic node to log into during deployment.

Create a password to log into the Ironic node:

```
openssl passwd -1 -stdin <<< YOUR_PASSWORD | sed 's/\$/\$\$/g'
```

Set debugging options on the IPA kernel parameters in `/etc/openstack_deploy/user_variables.yml`, substituing the encrypted password just generated into the `rootpwd` field. Ensure that the encrypted password is enclosed in double quotation marks.

```
ironic_ironic_conf_overrides:
  # Set a password on the root user in IPA for debug purposes
  pxe:
    kernel_append_params: 'ipa-debug=1 systemd.journald.forward_to_
↪console=yes rootpwd="<password-string>"'
```

**Note:** You must combine this override with any existing definition of `ironic_ironic_conf_overrides`.

Deploy the configuration file changes to the Ironic control plane.

The node can now be cleaned or provisioned, possibly pausing the deployment by enabling maintainance on the node and then logging into the node with SSH as the root user with the password previously encrypted.

This is an OpenStack-Ansible role to deploy the Bare Metal (ironic) service. See the role-ironic spec for more information.

To clone or view the source code for this repository, visit the role repository for os_ironic.

CHAPTER

SIX

# DEFAULT VARIABLES

```
# Defaults file for openstack-ansible-ironic

# Verbosity Options
debug: False

#python venv executable
ironic_venv_python_executable: "{{ openstack_venv_python_executable | default(
↪'python3') }}"

# Set the host which will execute the shade modules
# for the service setup. The host must already have
# clouds.yaml properly configured.
ironic_service_setup_host: "{{ openstack_service_setup_host | default(
↪'localhost') }}"
ironic_service_setup_host_python_interpreter: "{{ openstack_service_setup_
↪host_python_interpreter | default((ironic_service_setup_host == 'localhost
↪') | ternary(ansible_playbook_python, ansible_facts['python']['executable
↪'])) }}"

# Set the package install state for distribution packages
# Options are 'present' and 'latest'
ironic_package_state: "{{ package_state | default('latest') }}"

ironic_git_repo: https://opendev.org/openstack/ironic
ironic_inspector_git_repo: https://opendev.org/openstack/ironic-inspector
ironic_git_install_branch: master
ironic_inspector_git_install_branch: master
ironic_upper_constraints_url: "{{ requirements_git_url | default('https://
↪releases.openstack.org/constraints/upper/' ~ requirements_git_install_
↪branch | default('master')) }}"
ironic_git_constraints:
  - "--constraint {{ ironic_upper_constraints_url }}"

ironic_pip_install_args: "{{ pip_install_options | default('') }}"

# Name of the virtual env to deploy into
ironic_venv_tag: "{{ venv_tag | default('untagged') }}"
ironic_bin: "/openstack/venvs/ironic-{{ ironic_venv_tag }}/bin"
```

(continues on next page)

(continued from previous page)

```
# System info
ironic_system_user_name: ironic
ironic_system_group_name: ironic
ironic_system_shell: /bin/bash
ironic_system_comment: ironic system user
ironic_system_home_folder: "/var/lib/{{ ironic_system_user_name }}"
ironic_system_slice_name: ironic
ironic_lock_dir: "{{ openstack_lock_dir | default('/run/lock') }}"

# Ironic Program and Service names
python_ironic_client_program_name: ironic
ironic_services:
  ironic-api:
    group: ironic_api
    service_name: ironic-api
    init_config_overrides: "{{ ironic_api_init_config_overrides }}"
    wsgi_app: True
    wsgi_name: ironic-api-wsgi
    uwsgi_overrides: "{{ ironic_api_uwsgi_ini_overrides }}"
    uwsgi_port: "{{ ironic_service_port }}"
    uwsgi_bind_address: "{{ ironic_uwsgi_bind_address }}"
    uwsgi_tls: "{{ ironic_backend_ssl | ternary(ironic_uwsgi_tls, {}) }}"
  ironic-conductor:
    group: ironic_conductor
    service_name: ironic-conductor
    init_config_overrides: "{{ ironic_conductor_init_config_overrides }}"
    execstarts: "{{ ironic_bin }}/ironic-conductor"
  ironic-inspector:
    group: ironic_inspector
    service_name: ironic-inspector
    init_config_overrides: "{{ ironic_inspector_init_config_overrides }}"
    execstarts: "{{ ironic_bin }}/ironic-inspector"
  ironic-inspector-dnsmasq:
    group: ironic_inspector
    service_name: ironic-inspector-dnsmasq
    service_type: forking
    systemd_user_name: root
    systemd_group_name: root
    init_config_overrides: "{{ ironic_inspector_dnsmasq_init_config_overrides␣
↪}}"
    execstarts: "/usr/sbin/dnsmasq --conf-file=/etc/ironic-inspector/
↪inspector-dnsmasq.conf"
    after_targets:
      - openvswitch.service
      - network.target
    state: stopped

ironic_service_name: ironic
```

(continues on next page)

(continued from previous page)

```
ironic_service_type: baremetal
ironic_service_proto: http
ironic_service_publicuri_proto: "{{ openstack_service_publicuri_proto |␣
↪default(ironic_service_proto) }}"
ironic_service_adminuri_proto: "{{ openstack_service_adminuri_proto |␣
↪default(ironic_service_proto) }}"
ironic_service_internaluri_proto: "{{ openstack_service_internaluri_proto |␣
↪default(ironic_service_proto) }}"
ironic_service_port: 6385
ironic_service_description: "Ironic baremetal provisioning service"
ironic_service_publicuri: "{{ ironic_service_publicuri_proto }}://{{ external_
↪lb_vip_address }}:{{ ironic_service_port }}"
ironic_service_publicurl: "{{ ironic_service_publicuri }}"
ironic_service_adminuri: "{{ ironic_service_adminuri_proto }}://{{ internal_
↪lb_vip_address }}:{{ ironic_service_port }}"
ironic_service_adminurl: "{{ ironic_service_adminuri }}"
ironic_service_internaluri: "{{ ironic_service_internaluri_proto }}://{{␣
↪internal_lb_vip_address }}:{{ ironic_service_port }}"
ironic_service_internalurl: "{{ ironic_service_internaluri }}"
ironic_program_name: ironic-api
ironic_service_region: "{{ service_region | default('RegionOne') }}"
ironic_service_project_name: "service"
ironic_service_project_domain_id: default
ironic_service_user_domain_id: default
ironic_service_role_names:
  - admin
  - service
ironic_service_token_roles:
  - service
ironic_service_token_roles_required: "{{ openstack_service_token_roles_
↪required | default(True) }}"
ironic_service_in_ldap: "{{ service_ldap_backend_enabled | default(False) }}"

# The name of the entry in container_networks for the bmaas network
# This is the default provisioning / inspection / cleaning network for this␣
↪role
ironic_container_network_name: "bmaas_address"

# The name of the bridge on the host for the bmaas network
ironic_bmaas_bridge: "{{ container_networks[ironic_container_network_name][
↪'bridge'] | default('bridge_undefined') }}"

# The address of this host on the bmaas network
ironic_bmaas_address: "{{ (is_metal | default(False)) | ternary(ansible_
↪facts[ironic_bmaas_bridge | replace('-','_')]['ipv4']['address'],
                                                                container_
↪networks[ironic_container_network_name]['address']) | default('address_
↪undefined') }}"
# The name of the interface on the bmaas network
```

(continues on next page)

(continued from previous page)

```
# This is the bmaas bridge name on metal, or the corresponding interface name␣
↪in a container
ironic_bmaas_interface: "{{ (is_metal | default(False)) | ternary(ironic_
↪bmaas_bridge,
                                                                  container_
↪networks[ironic_container_network_name]['interface']) | default('interface_
↪undefined') }}"

# Ironic image store information
#
### Hosted Web Server
#
# Set this to True to use http web server to host floppy
# images and generated boot ISO. This requires http_root and
# http_url to be configured in the [deploy] section of the
# config file. If this is set to False, then Ironic will use
# Swift to host the floppy images and generated boot_iso.
ironic_enable_web_server_for_images: False
ironic_http_bind_address: "{{ ironic_bmaas_address }}"
ironic_http_url: "{{ ironic_ipxe_proto }}://{{ ironic_http_bind_address }}:{{␣
↪ironic_ipxe_port }}"
ironic_http_root: "/httpboot"
#
### Swift Config
#
ironic_swift_image_container: glance_images
ironic_swift_api_version: v1
ironic_swift_url_endpoint_type: swift
# The ironic swift auth account and swift endpoints will be generated using␣
↪the
#  known swift data as provided by swift stat. If you wish to set either of␣
↪these
#  items to something else define these variables.
# ironic_swift_auth_account: AUTH_1234567890
# ironic_swift_endpoint: https://localhost:8080

# Is this Ironic installation working standalone?
# If you're wanting Ironic to work without being integrated to other OpenStack
# services, set this to True, and update the dhcp configuration appropriately
ironic_standalone: False

# Enables or disables automated cleaning. Automated cleaning
# is a configurable set of steps, such as erasing disk drives,
# that are performed on the node to ensure it is in a baseline
# state and ready to be deployed to.
ironic_automated_clean: false
# Set to 0 to disable erase devices on cleaning
ironic_erase_devices_priority: 10
```

(continues on next page)

(continued from previous page)

```
# Database
ironic_db_setup_host: "{{ openstack_db_setup_host | default('localhost') }}"
ironic_db_setup_python_interpreter: "{{ openstack_db_setup_python_interpreter␣
↪| default((ironic_db_setup_host == 'localhost') | ternary(ansible_playbook_
↪python, ansible_facts['python']['executable'])) }}"
ironic_galera_address: "{{ galera_address | default('127.0.0.1') }}"
ironic_galera_user: ironic
ironic_galera_database: ironic
ironic_galera_use_ssl: "{{ galera_use_ssl | default(False) }}"
ironic_galera_ssl_ca_cert: "{{ galera_ssl_ca_cert | default('') }}"
ironic_galera_port: "{{ galera_port | default('3306') }}"
ironic_db_max_overflow: "{{ openstack_db_max_overflow | default('50') }}"
ironic_db_max_pool_size: "{{ openstack_db_max_pool_size | default('5') }}"
ironic_db_pool_timeout: "{{ openstack_db_pool_timeout | default('30') }}"
ironic_db_connection_recycle_time: "{{ openstack_db_connection_recycle_time |␣
↪default('600') }}"

## Keystone authentication middleware
ironic_keystone_auth_plugin: password

# Neutron network - Set these in a playbook/task - can be set manually.
# Only "name" or "uuid" is needed, uuid will take preference if both are␣
↪specified.
# The cleaning and inspection network is not required to be set; they will␣
↪default
# to the provisioning network if not specified.
# ironic_neutron_provisioning_network_uuid: "UUID for provisioning network in␣
↪neutron"
# ironic_neutron_cleaning_network_uuid: "UUID for cleaning network in neutron"
# ironic_neutron_inspection_network_uuid: "UUID for inspection network in␣
↪neutron"
# ironic_neutron_provisioning_network_name: "Name of provisioning network in␣
↪neutron"
# ironic_neutron_cleaning_network_name: "Name of cleaning network in neutron"
# ironic_neutron_inspection_network_name: "Name of inspection network in␣
↪neutron"

# Integrated Openstack configuration
ironic_enabled_network_interfaces_list: "flat,noop{{ (ironic_neutron_
↪provisioning_network_uuid is defined) | ternary(',neutron','') }}"
ironic_default_network_interface: "{{ (ironic_neutron_provisioning_network_
↪uuid is defined) | ternary('neutron','flat') }}"
ironic_auth_strategy: keystone
ironic_dhcp_provider: "{{ (ironic_standalone | bool) | ternary('none',
↪'neutron') }}"
ironic_sync_power_state_interval: "{{ (ironic_standalone | bool) | ternary('-1
↪', '60') }}"
ironic_db_connection_string: "mysql+pymysql://{{ ironic_galera_user }}:{{␣
↪ironic_container_mysql_password }}@{{ ironic_galera_address }}:{{ ironic_
↪galera_port }}/ironic?charset=utf8{% if ironic_galera_use_ssl | bool %}&ssl_
↪verify_cert=true{% if ironic_galera_ssl_ca_cert | length > 0 %}&ssl_ca={{␣
↪ironic_galera_ssl_ca_cert }}{% endif %}{% endif %}"
```

(continues on next page)

(continued from previous page)

```
# Common configuration
ironic_node_name: ironic

# If you want to regenerate the ironic users SSH keys, on each run, set this
# var to True. Otherwise keys will be generated on the first run and not
# regenerated each run.
ironic_recreate_keys: False

ironic_tftp_server_address: "{{ ironic_bmaas_address }}"

# Use this variable to add extra files into the ironic_tftp_root directory
# ironic_tftp_extra_content:
#   - path: /some/local/dir/local-file.txt
#     name: local-file.txt
#   - url: http://boot.ipxe.org/arm64-efi/ipxe.efi
#     name: ipxe_aa64.efi
ironic_tftp_extra_content: []

ironic_pip_packages:
  - "git+{{ ironic_git_repo }}@{{ ironic_git_install_branch }}#egg=ironic"
  - cryptography
  - osprofiler
  - proliantutils
  - PyMySQL
  - pymemcache
  - pysnmp
  - python-dracclient
  - python-ilorest-library
  - python-ironicclient
  - python-memcached
  - python-scciclient
  - python-swiftclient
  - python-xclarityclient
  - sushy
  - systemd-python

# ipmitool-socat console settings
ironic_socat_bind_address: "{{ openstack_service_bind_address | default('0.0.
↪0.0') }}"
ironic_socat_port_range: "10000:10099"

# Specific pip packages provided by the user for the ironic service
ironic_user_pip_packages: []

ironic_inspector_pip_packages:
  - "git+{{ ironic_inspector_git_repo }}@{{ ironic_inspector_git_install_
↪branch }}#egg=ironic-inspector"
  - python-ironic-inspector-client
```

(continues on next page)

(continued from previous page)

```
# Specific pip packages provided by the user for the ironic inspector service
ironic_inspector_user_pip_packages: []

# Memcached override
ironic_memcached_servers: "{{ memcached_servers }}"

## Oslo Messaging Info
# RPC
ironic_oslomsg_rpc_host_group: "{{ oslomsg_rpc_host_group | default('rabbitmq_
↪all') }}"
ironic_oslomsg_rpc_setup_host: "{{ (ironic_oslomsg_rpc_host_group in groups)␣
↪| ternary(groups[ironic_oslomsg_rpc_host_group][0], 'localhost') }}"
ironic_oslomsg_rpc_transport: "{{ oslomsg_rpc_transport | default('rabbit') }}
↪"
ironic_oslomsg_rpc_servers: "{{ oslomsg_rpc_servers | default('127.0.0.1') }}"
ironic_oslomsg_rpc_port: "{{ oslomsg_rpc_port | default('5672') }}"
ironic_oslomsg_rpc_use_ssl: "{{ oslomsg_rpc_use_ssl | default(False) }}"
ironic_oslomsg_rpc_userid: ironic
ironic_oslomsg_rpc_vhost: /ironic
ironic_oslomsg_rpc_ssl_version: "{{ oslomsg_rpc_ssl_version | default('TLSv1_2
↪') }}"
ironic_oslomsg_rpc_ssl_ca_file: "{{ oslomsg_rpc_ssl_ca_file | default('') }}"

# Notify
ironic_oslomsg_notify_host_group: "{{ oslomsg_notify_host_group | default(
↪'rabbitmq_all') }}"
ironic_oslomsg_notify_setup_host: "{{ (ironic_oslomsg_notify_host_group in␣
↪groups) | ternary(groups[ironic_oslomsg_notify_host_group][0], 'localhost')␣
↪}}"
ironic_oslomsg_notify_transport: "{{ oslomsg_notify_transport | default(
↪'rabbit') }}"
ironic_oslomsg_notify_servers: "{{ oslomsg_notify_servers | default('127.0.0.1
↪') }}"
ironic_oslomsg_notify_port: "{{ oslomsg_notify_port | default('5672') }}"
ironic_oslomsg_notify_use_ssl: "{{ oslomsg_notify_use_ssl | default(False) }}"
ironic_oslomsg_notify_userid: "{{ ironic_oslomsg_rpc_userid }}"
ironic_oslomsg_notify_password: "{{ ironic_oslomsg_rpc_password }}"
ironic_oslomsg_notify_vhost: "{{ ironic_oslomsg_rpc_vhost }}"
ironic_oslomsg_notify_ssl_version: "{{ oslomsg_notify_ssl_version | default(
↪'TLSv1_2') }}"
ironic_oslomsg_notify_ssl_ca_file: "{{ oslomsg_notify_ssl_ca_file | default('
↪') }}"

## (Qdrouterd) integration
# TODO(ansmith): Change structure when more backends will be supported
ironic_oslomsg_amqp1_enabled: "{{ ironic_oslomsg_rpc_transport == 'amqp' }}"

ironic_optional_oslomsg_amqp1_pip_packages:
```

(continues on next page)

(continued from previous page)

```
  - oslo.messaging[amqp1]

# Auth
ironic_service_user_name: "ironic"

# WSGI settings
ironic_wsgi_threads: 1
ironic_wsgi_processes_max: 16
ironic_wsgi_processes: "{{ [[(ansible_facts['processor_vcpus']//ansible_facts[
↪'processor_threads_per_core'])|default(1), 1] | max * 2, ironic_wsgi_
↪processes_max] | min }}"
ironic_uwsgi_bind_address: "{{ openstack_service_bind_address | default('0.0.
↪0.0') }}"
ironic_uwsgi_tls:
  crt: "{{ ironic_ssl_cert }}"
  key: "{{ ironic_ssl_key }}"

### OpenStack Services to integrate with

# Glance
ironic_glance_auth_strategy: "{{ ironic_auth_strategy }}"
ironic_glance_service_project_name: "{{ glance_service_project_name | default(
↪'service') }}"
ironic_glance_service_project_domain_id: "{{ glance_service_project_domain_id␣
↪| default('default') }}"
ironic_glance_keystone_auth_plugin: "{{ glance_keystone_auth_plugin | default(
↪'password') }}"
ironic_glance_service_user_name: "{{ glance_service_user_name | default(
↪'glance') }}"
ironic_glance_service_user_domain_id: "{{ glance_service_user_domain_id |␣
↪default('default') }}"
ironic_glance_keystone_auth_url: "{{ keystone_service_internalurl | default(
↪'http://localhost:5000/v3') }}"

# Neutron
ironic_neutron_auth_strategy: "{{ ironic_auth_strategy }}"

### Config Overrides
ironic_ironic_conf_overrides: {}
ironic_rootwrap_conf_overrides: {}
ironic_policy_overrides: {}
ironic_api_uwsgi_ini_overrides: {}

# pxe boot
ironic_kernel_append_params: "ipa-debug=1 systemd.journald.forward_to_
↪console=yes"

ironic_api_init_config_overrides: {}
ironic_conductor_init_config_overrides: {}
```

(continues on next page)

(continued from previous page)

```
# driver definitions
ironic_drivers_enabled:
  - no_driver
  - agent_ipmitool
  - pxe_ipmitool

ironic_inspector_developer_mode: false
ironic_inspector_venv_python_executable: "{{ openstack_venv_python_executable␣
↪| default('python2') }}"

# System info
ironic_inspector_service_setup_host: "{{ openstack_service_setup_host |␣
↪default('localhost') }}"
ironic_inspector_service_name: ironic-inspector
ironic_inspector_service_type: baremetal-introspection
ironic_inspector_service_description: "Ironic Baremetal Introspection Service"
ironic_inspector_service_publicuri_proto: "{{ openstack_service_publicuri_
↪proto | default(ironic_service_proto) }}"
ironic_inspector_service_adminuri_proto: "{{ openstack_service_adminuri_proto␣
↪| default(ironic_service_proto) }}"
ironic_inspector_service_internaluri_proto: "{{ openstack_service_internaluri_
↪proto | default(ironic_service_proto) }}"
ironic_inspector_service_address: "{{ openstack_service_bind_address }}"
ironic_inspector_service_port: 5050
ironic_inspector_service_publicuri: "{{ ironic_inspector_service_publicuri_
↪proto }}://{{ external_lb_vip_address }}:{{ ironic_inspector_service_port }}
↪"
ironic_inspector_service_publicurl: "{{ ironic_inspector_service_publicuri }}"
ironic_inspector_service_adminuri: "{{ ironic_inspector_service_adminuri_
↪proto }}://{{ internal_lb_vip_address }}:{{ ironic_inspector_service_port }}
↪"
ironic_inspector_service_adminurl: "{{ ironic_inspector_service_adminuri }}"
ironic_inspector_service_internaluri: "{{ ironic_inspector_service_
↪internaluri_proto }}://{{ internal_lb_vip_address }}:{{ ironic_inspector_
↪service_port }}"
ironic_inspector_service_internalurl: "{{ ironic_inspector_service_
↪internaluri }}"
ironic_inspector_service_role_names:
  - admin
  - service
ironic_inspector_service_token_roles:
  - service
ironic_inspector_service_token_roles_required: "{{ openstack_service_token_
↪roles_required | default(True) }}"
ironic_inspector_service_project_name: "service"
ironic_inspector_service_in_ldap: "{{ service_ldap_backend_enabled |␣
↪default(False) }}"
ironic_inspector_service_domain_id: default
```

(continues on next page)

(continued from previous page)

```
ironic_inspector_callback_url: "{{ ironic_inspector_service_internaluri_proto␣
↪}}://{{ internal_lb_vip_address }}:{{ ironic_inspector_service_port }}/v1/
↪continue"

# Database
ironic_inspector_db_setup_host: "{{ openstack_db_setup_host | default(
↪'localhost') }}"
ironic_inspector_db_setup_python_interpreter: "{{ openstack_db_setup_python_
↪interpreter | default((ironic_inspector_db_setup_host == 'localhost') |␣
↪ternary(ansible_playbook_python, ansible_facts['python']['executable'])) }}"
ironic_inspector_galera_address: "{{ galera_address | default('127.0.0.1') }}"
ironic_inspector_galera_user: ironic-inspector
ironic_inspector_galera_database: ironic_inspector
ironic_inspector_galera_port: 3306
ironic_inspector_galera_use_ssl: "{{ galera_use_ssl | default(False) }}"
ironic_inspector_galera_ssl_ca_cert: "{{ galera_ssl_ca_cert | default('') }}"
ironic_inspector_db_max_overflow: "{{ openstack_db_max_overflow | default('50
↪') }}"
ironic_inspector_db_max_pool_size: "{{ openstack_db_max_pool_size | default('5
↪') }}"
ironic_inspector_db_pool_timeout: "{{ openstack_db_pool_timeout | default('30
↪') }}"
ironic_inspector_db_connection_recycle_time: "{{ openstack_db_connection_
↪recycle_time | default('600') }}"

ironic_inspector_pip_install_args: "{{ pip_install_options | default('') }}"

# Ironic iPXE support
ironic_ipxe_enabled: False
ironic_ipxe_port: 8051
ironic_ipxe_proto: "http"

# Auth
ironic_inspector_service_user_name: "ironic_inspector"

### OpenStack Services to integrate with
# Ironic swift store information
ironic_inspector_swift_user_name: swift-inspector
ironic_inspector_swift_role_names:
  - member
  - swiftoperator

#Ironic deploy images need to be uploaded to glance.
ironic_deploy_image_glance_upload: True

# Set the directory where the downloaded image will be stored
# on the ironic_service_setup_host host. If the host is localhost,
# then the user running the playbook must have access to it.
ironic_deploy_image_path: "/root/openstack-ansible/ironic"
```

(continues on next page)

(continued from previous page)

```
ironic_deploy_image_path_owner: "root"

#The default download URL is like https://tarballs.opendev.org/openstack/
↪ironic-python-agent/dib/files/ipa-centos8-stable-xena.initramfs
#Allow various parts of this to be overidden to local mirrors, or replaced␣
↪completely with custom settings
ironic_deploy_image_server: "https://tarballs.opendev.org/"
ironic_deploy_image_server_path: "openstack/ironic-python-agent/dib/files/"
ironic_deploy_image_base_name: "ipa-centos9-stable-2023.1"
ironic_deploy_image_kernel_name: "{{ ironic_deploy_image_base_name + '.kernel
↪' }}"
ironic_deploy_image_initramfs_name: "{{ ironic_deploy_image_base_name + '.
↪initramfs' }}"
ironic_deploy_image_list:
 - url: "{{ ironic_deploy_image_server ~ ironic_deploy_image_server_path ~␣
↪ironic_deploy_image_kernel_name }}"
   sha_url: "{{ ironic_deploy_image_server ~ ironic_deploy_image_server_path ~
↪ ironic_deploy_image_kernel_name ~ '.sha256' }}"
   container_format: 'aki'
   disk_format: 'aki'
   name: "{{ ironic_deploy_image_kernel_name }}"
 - url: "{{ ironic_deploy_image_server ~ ironic_deploy_image_server_path ~␣
↪ironic_deploy_image_initramfs_name }}"
   sha_url: "{{ ironic_deploy_image_server ~ ironic_deploy_image_server_path ~
↪ ironic_deploy_image_initramfs_name ~ '.sha256' }}"
   container_format: 'ari'
   disk_format: 'ari'
   name: "{{ ironic_deploy_image_initramfs_name }}"

#allow user defined extra images to upload
ironic_extra_deploy_image_list: []

# Ironic inspector
ironic_inspector_enable_discovery: True
ironic_inspector_openstack_db_connection_string: "mysql+pymysql://{{ ironic_
↪inspector_galera_user }}:{{ ironic_inspector_container_mysql_password }}@{{␣
↪ironic_inspector_galera_address }}:{{ ironic_inspector_galera_port }}/{{␣
↪ironic_inspector_galera_database }}?charset=utf8{% if ironic_inspector_
↪galera_use_ssl | bool %}&ssl_verify_cert=true{% if ironic_inspector_galera_
↪ssl_ca_cert | length > 0 %}&ssl_ca={{ ironic_inspector_galera_ssl_ca_cert }}
↪{% endif %}{% endif %}"

#define this to adjust the inspector processing hooks
#Example:
#ironic_inspector_processing_hooks: "$default_processing_hooks,lldp_basic,
↪local_link_connection"

#pass additional kernel paramters to the deploy image
ironic_inspector_extra_callback_parameters: ''
```

(continues on next page)

(continued from previous page)

```
# Ironic inspector dhcp
ironic_inspector_dhcp_address: "{{ ironic_bmaas_address }}"
ironic_inspector_dhcp_pool_range: 192.168.0.51 192.168.0.150
ironic_inspector_dhcp_subnet: 192.168.0.0/22
ironic_inspector_dhcp_subnet_mask: 255.255.252.0
ironic_insepctor_dhcp_enable_gateway: True
ironic_inspector_dhcp_gateway: 192.168.0.1
ironic_inspector_dhcp_enable_nameservers: True
ironic_inspector_dhcp_nameservers: 192.168.0.1
ironic_inspector_dhcp_lease_time: 600

ironic_inspector_dhcp_type: dnsmasq # isc_dhcp
ironic_inspector_boot_mode: http #tftp
ironic_inspector_pxe_boot_mode: "{{ ironic_inspector_boot_mode }}"
ironic_inspector_httpboot_dir: "{{ ironic_http_root }}"
ironic_inspector_tftpboot_dir: "{{ ironic_tftpd_root }}"

ironic_inspector_dhcp_interface: "{{ ironic_bmaas_interface }}"
ironic_inspector_valid_interfaces: internal,public

### Config Overrides
ironic_inspector_conf_overrides: {}
ironic_inspector_rootwrap_conf_overrides: {}
ironic_inspector_init_config_overrides: {}
ironic_inspector_dnsmasq_init_config_overrides: {}
# pxe boot
ironic_inspector_pxe_append_params: "ipa-debug=1 systemd.journald.forward_to_
↪console=yes" #ipa-inspection-collectors=default,logs,extra_hardware

ironic_inspector_pxe_filter: dnsmasq #iptables

ironic_inspector_oslomsg_rpc_host_group: "{{ oslomsg_rpc_host_group | default(
↪'rabbitmq_all') }}"
ironic_inspector_oslomsg_rpc_setup_host: "{{ (ironic_oslomsg_rpc_host_group␣
↪in groups) | ternary(groups[ironic_oslomsg_rpc_host_group][0], 'localhost')␣
↪}}"
ironic_inspector_oslomsg_rpc_transport: "{{ oslomsg_rpc_transport | default(
↪'rabbit') }}"
ironic_inspector_oslomsg_rpc_servers: "{{ oslomsg_rpc_servers | default('127.
↪0.0.1') }}"
ironic_inspector_oslomsg_rpc_port: "{{ oslomsg_rpc_port | default('5672') }}"
ironic_inspector_oslomsg_rpc_use_ssl: "True"
ironic_inspector_oslomsg_rpc_userid: ironic
ironic_inspector_oslomsg_rpc_vhost: /ironic
ironic_inspector_oslomsg_rpc_ssl_version: "{{ oslomsg_rpc_ssl_version |␣
↪default('TLSv1_2') }}"
ironic_inspector_oslomsg_rpc_ssl_ca_file: "{{ oslomsg_rpc_ssl_ca_file |␣
↪default('') }}"
```

(continues on next page)

(continued from previous page)

```
ironic_inspector_oslomsg_notify_host_group: "{{ oslomsg_notify_host_group |␣
↪default('rabbitmq_all') }}"
ironic_inspector_oslomsg_notify_setup_host: "{{ (ironic_inspector_oslomsg_
↪notify_host_group in groups) | ternary(groups[ironic_inspector_oslomsg_
↪notify_host_group][0], 'localhost') }}"
ironic_inspector_oslomsg_notify_transport: "{{ oslomsg_notify_transport |␣
↪default('rabbit') }}"
ironic_inspector_oslomsg_notify_servers: "{{ oslomsg_notify_servers | default(
↪'127.0.0.1') }}"
ironic_inspector_oslomsg_notify_port: "{{ oslomsg_notify_port | default('5672
↪') }}"
ironic_inspector_oslomsg_notify_use_ssl: "False"
ironic_inspector_oslomsg_notify_userid: "{{ ironic_inspector_oslomsg_rpc_
↪userid }}"
ironic_inspector_oslomsg_notify_password: "{{ ironic_oslomsg_rpc_password }}"
ironic_inspector_oslomsg_notify_vhost: "{{ ironic_inspector_oslomsg_rpc_vhost␣
↪}}"
ironic_inspector_oslomsg_notify_ssl_version: "{{ oslomsg_notify_ssl_version |␣
↪default('TLSv1_2') }}"
ironic_inspector_oslomsg_notify_ssl_ca_file: "{{ oslomsg_notify_ssl_ca_file |␣
↪default('') }}"

ironic_inspector_optional_oslomsg_amqp1_pip_packages:
  - oslo.messaging[amqp1]
ironic_inspector_oslomsg_amqp1_enabled: True

###
### Backend TLS
###

# Define if communication between haproxy and service backends should be
# encrypted with TLS.
ironic_backend_ssl: "{{ openstack_service_backend_ssl | default(False) }}"

# Storage location for SSL certificate authority
ironic_pki_dir: "{{ openstack_pki_dir | default('/etc/openstack_deploy/pki') }
↪}"

# Delegated host for operating the certificate authority
ironic_pki_setup_host: "{{ openstack_pki_setup_host | default('localhost') }}"

# ironic server certificate
ironic_pki_keys_path: "{{ ironic_pki_dir ~ '/certs/private/' }}"
ironic_pki_certs_path: "{{ ironic_pki_dir ~ '/certs/certs/' }}"
ironic_pki_intermediate_cert_name: "{{ openstack_pki_service_intermediate_
↪cert_name | default('ExampleCorpIntermediate') }}"
ironic_pki_regen_cert: ''
ironic_pki_san: "{{ openstack_pki_san | default('DNS:' ~ ansible_facts[
↪'hostname'] ~ ',IP:' ~ management_address) }}"
```

(continues on next page)

(continued from previous page)

```
ironic_pki_certificates:
  - name: "ironic_{{ ansible_facts['hostname'] }}"
    provider: ownca
    cn: "{{ ansible_facts['hostname'] }}"
    san: "{{ ironic_pki_san }}"
    signed_by: "{{ ironic_pki_intermediate_cert_name }}"

# ironic destination files for SSL certificates
ironic_ssl_cert: /etc/ironic/ironic.pem
ironic_ssl_key: /etc/ironic/ironic.key

# Installation details for SSL certificates
ironic_pki_install_certificates:
  - src: "{{ ironic_user_ssl_cert | default(ironic_pki_certs_path ~ 'ironic_'␣
↪~ ansible_facts['hostname'] ~ '-chain.crt') }}"
    dest: "{{ ironic_ssl_cert }}"
    owner: "{{ ironic_system_user_name }}"
    group: "{{ ironic_system_user_name }}"
    mode: "0644"
  - src: "{{ ironic_user_ssl_key | default(ironic_pki_keys_path ~ 'ironic_' ~␣
↪ansible_facts['hostname'] ~ '.key.pem') }}"
    dest: "{{ ironic_ssl_key }}"
    owner: "{{ ironic_system_user_name }}"
    group: "{{ ironic_system_user_name }}"
    mode: "0600"

# Define user-provided SSL certificates
#ironic_user_ssl_cert: <path to cert on ansible deployment host>
#ironic_user_ssl_key: <path to cert on ansible deployment host>
```

CHAPTER

SEVEN

# DEPENDENCIES

This role needs pip >= 7.1 installed on the target host.

CHAPTER

EIGHT

# EXAMPLE PLAYBOOK

```
---
- name: Playbook for role testing
  hosts: localhost
  remote_user: root
  roles:
    - role: "os_ironic"
  vars:
    galera_root_user: root
  vars_prompt:
    - name: "galera_root_password"
      prompt: "What is galera_root_password?"
```

CHAPTER

NINE

# TAGS

This role supports the `ironic-install` and `ironic-config` tags. Use the `ironic-install` tag to install and upgrade. Use the `ironic-config` tag to maintain configuration of the service.