kayobe Documentation

Release

OpenStack Foundation

Sep 10, 2017

Contents

CHAPTER 1

Kayobe

Deployment of Scientific OpenStack using OpenStack kolla.

Kayobe is an open source tool for automating deployment of Scientific OpenStack onto a set of bare metal servers. Kayobe is composed of Ansible playbooks, a python module, and makes heavy use of the OpenStack kolla project. Kayobe aims to complement the kolla-ansible project, providing an opinionated yet highly configurable OpenStack deployment and automation of many operational procedures.

- Documentation: https://github.com/stackhpc/kayobe/tree/master/doc
- Source: https://github.com/stackhpc/kayobe
- Bugs: https://github.com/stackhpc/kayobe/issues

Features

- Heavily automated using Ansible
- *kayobe* Command Line Interface (CLI) for cloud operators
- Deployment of a *seed* VM used to manage the OpenStack control plane
- Configuration of physical network infrastructure
- Discovery, introspection and provisioning of control plane hardware using OpenStack bifrost
- Deployment of an OpenStack control plane using OpenStack kolla-ansible
- Discovery, introspection and provisioning of bare metal compute hosts using OpenStack ironic and ironic inspector
- Containerised workloads on bare metal using OpenStack magnum
- Big data on bare metal using OpenStack sahara

In the near future we aim to add support for the following:

- Control plane and workload monitoring and log aggregation using OpenStack monasca

- Virtualised compute using OpenStack nova

Documentation

Note: Kayobe and its documentation is currently under heavy development, and therefore may be incomplete or out of date. If in doubt, contact the project's maintainers.

Architecture

Hosts in the System

In a system deployed by Kayobe we define a number of classes of hosts.

Control host The control host is the host on which kayobe, kolla and kolla-ansible will be installed, and is typically where the cloud will be managed from.

Seed host The seed host runs the bifrost deploy container and is used to provision the cloud hosts. Typically the seed host is deployed as a VM but this is not mandatory.

Cloud hosts The cloud hosts run the OpenStack control plane, storage, and virtualised compute services. Typically the cloud hosts run on bare metal but this is not mandatory.

Bare metal compute hosts: In a cloud providing bare metal compute services to tenants via ironic, these hosts will run the bare metal tenant workloads. In a cloud with only virtualised compute this category of hosts does not exist.

Note: In many cases the control and seed host will be the same, although this is not mandatory.

Networks

Kayobe's network configuration is very flexible but does define a few default classes of networks. These are logical networks and may map to one or more physical networks in the system.

Overcloud provisioning network The overcloud provisioning network is used by the seed host to provision the cloud hosts.

Workload provisioning network The workload provisioning network is used by the cloud hosts to provision the bare metal compute hosts.

Internal network The internal network hosts the internal and admin OpenStack API endpoints.

External network The external network hosts the public OpenStack API endpoints and provides external network access for the hosts in the system.

Installation

Prerequisites

Currently Kayobe supports the following Operating Systems:

- CentOS 7.3

To avoid conflicts with python packages installed by the system package manager it is recommended to install Kayobe in a virtualenv. Ensure that the `virtualenv` python module is available on the control host. For example, on CentOS:

```
$ yum install -y python-virtualenv
```

Installation

This guide will describe how to install Kayobe from source in a virtualenv. First, obtain the Kayobe source code. For example:

```
$ git clone https://github.com/stackhpc/kayobe
```

Create a virtualenv for Kayobe:

```
$ cd kayobe
$ virtualenv kayobe-venv
```

Activate the virtualenv and update pip:

```
$ source kayobe-venv/bin/activate
(kayobe-venv) $ pip install -U pip
```

Install Kayobe and its dependencies using the source code checkout:

```
(kayobe-venv) $ pip install .
```

Finally, deactivate the virtualenv:

```
(kayobe-venv) $ deactivate
```

Configuration

This section covers configuration of Kayobe. As an Ansible-based project, Kayobe is for the most part configured using YAML files.

Configuration Location

Kayobe configuration is by default located in `/etc/kayobe` on the Ansible control host. This location can be overridden to a different location to avoid touching the system configuration directory by setting the environment variable `KAYOBE_CONFIG_PATH`. Similarly, kolla configuration on the Ansible control host will by default be located in `/etc/kolla` and can be overridden via `KOLLA_CONFIG_PATH`.

Configuration Directory Layout

The Kayobe configuration directory contains Ansible `extra-vars` files and the Ansible inventory. An example of the directory structure is as follows:

```
extra-vars1.yml
extra-vars2.yml
inventory/
    group_vars/
```

```
        group1-vars
        group2-vars
    groups
    host_vars/
        host1-vars
        host2-vars
    hosts
```

Configuration Patterns

Ansible's variable precedence rules are fairly well documented and provide a mechanism we can use for providing site localisation and customisation of OpenStack in combination with some reasonable default values. For global configuration options, Kayobe typically uses the following patterns:

- Playbook group variables for the *all* group in `<kayobe repo>/ansible/group_vars/all/*` set **global defaults**. These files should not be modified.
- Playbook group variables for other groups in `<kayobe repo>/ansible/group_vars/<group>/*` set **defaults for some subsets of hosts**. These files should not be modified.
- Extra-vars files in `${KAYOBE_CONFIG_PATH}/*.yml` set **custom values for global variables** and should be used to apply global site localisation and customisation. By default these variables are commented out.

Additionally, variables can be set on a per-host basis using inventory host variables files in `${KAYOBE_CONFIG_PATH}/inventory/host_vars/*`. It should be noted that variables set in extra-vars files take precedence over per-host variables.

Configuring Kayobe

The kayobe-config git repository contains a Kayobe configuration directory structure and unmodified configuration files. This repository can be used as a mechanism for version controlling Kayobe configuration. As Kayobe is updated, the configuration should be merged to incorporate any upstream changes with local modifications.

Alternatively, the baseline Kayobe configuration may be copied from a checkout of the Kayobe repository to the Kayobe configuration path:

```
$ cp -r etc/ ${KAYOBE_CONFIG_PATH:-/etc/kayobe}
```

Once in place, each of the YAML and inventory files should be manually inspected and configured as required.

Inventory

The inventory should contain the following hosts:

Control host This should be localhost and should be a member of the `config-mgmt` group.

Seed hypervisor If provisioning a seed VM, a host should exist for the hypervisor that will run the VM, and should be a member of the `seed-hypervisor` group.

Seed The seed host, whether provisioned as a VM by Kayobe or externally managed, should exist in the `seed` group.

Cloud hosts and bare metal compute hosts are not required to exist in the inventory.

Site Localisation and Customisation

Site localisation and customisation is applied using Ansible extra-vars files in `${KAYOBE_CONFIG_PATH}/*.yml`.

Encryption of Secrets

Kayobe supports the use of Ansible vault to encrypt sensitive information in its configuration. The `ansible-vault` tool should be used to manage individual files for which encryption is required. Any of the configuration files may be encrypted. Since encryption can make working with Kayobe difficult, it is recommended to follow best practice, adding a layer of indirection and using encryption only where necessary.

Network Configuration

Kayobe provides a flexible mechanism for configuring the networks in a system. Kayobe networks are assigned a name which is used as a prefix for variables that define the network's attributes. For example, to configure the `cidr` attribute of a network named `arpanet`, we would use a variable named `arpanet_cidr`.

Global network configuration is stored in `${KAYOBE_CONFIG_PATH}/networks.yml`. The following attributes are supported:

`cidr` CIDR representation (<IP>/<prefix length>) of the network's IP subnet.

`allocation_pool_start` IP address of the start of Kayobe's allocation pool range.

`allocation_pool_end` IP address of the end of Kayobe's allocation pool range.

`inspection_allocation_pool_start` IP address of the start of ironic inspector's allocation pool range.

`inspection_allocation_pool_end` IP address of the end of ironic inspector's allocation pool range.

`neutron_allocation_pool_start` IP address of the start of neutron's allocation pool range.

`neutron_allocation_pool_end` IP address of the end of neutron's allocation pool range.

`gateway` IP address of the network's default gateway.

`vlan` VLAN ID.

`mtu` Maximum Transmission Unit (MTU).

IP addresses are allocated automatically by Kayobe from the allocation pool defined by `allocation_pool_start` and `allocation_pool_end`. The allocated addresses are stored in `${KAYOBE_CONFIG_PATH}/network-allocation.yml` using the global per-network attribute `ips` which maps Ansible inventory hostnames to allocated IPs.

Some network attributes are specific to a host's role in the system, and these are stored in `${KAYOBE_CONFIG_PATH}/inventory/group_vars/<group>/network-interfaces`. The following attributes are supported:

`interface` The name of the network interface attached to the network.

`bridge_ports` For bridge interfaces, a list of names of network interfaces to add to the bridge.

In order to provide flexibility in the system's network topology, Kayobe maps the named networks to logical network roles. A single named network may perform multiple roles, or even none at all. The available roles are:

`provision_oc_net_name` Name of the network used by the seed to provision the bare metal overcloud hosts.

`provision_wl_net_name` Name of the network used by the overcloud hosts to provision the bare metal workload hosts.

internal_net_name Name of the network used to expose the internal OpenStack API endpoints.

external_net_name Name of the network used to expose the external OpenStack API endpoints and to provide external network access via Neutron.

storage_net_name Name of the network used to carry storage data traffic.

storage_mgmt_net_name Name of the network used to carry storage management traffic.

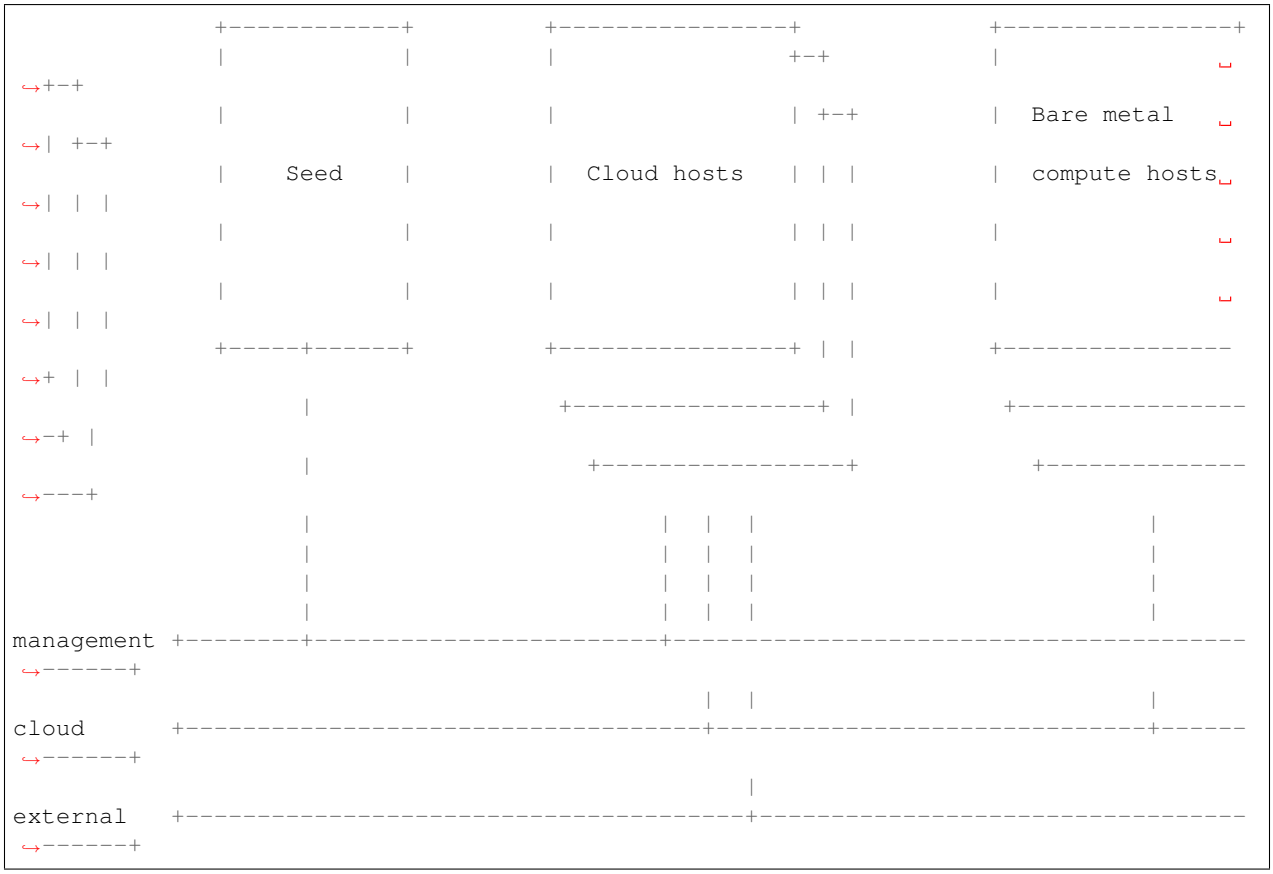
inspection_net_name Name of the network used to perform hardware introspection on the bare metal workload hosts.

These roles are configured in `${KAYOBE_CONFIG_PATH}/networks.yml`.

Networks are mapped to hosts using the variable `network_interfaces`. Kayobe's playbook group variables define some sensible defaults for this variable for hosts in the `seed` and `controllers` groups based on the logical network roles. These defaults can be extended by setting the variables `seed_extra_network_interfaces` and `controller_extra_network_interfaces` in `${KAYOBE_CONFIG_PATH}/seed.yml` and `${KAYOBE_CONFIG_PATH}/controllers.yml` respectively.

Example

In our example cloud we have three networks: `management`, `cloud` and `external`:



The `management` network is used to access the servers' BMCs and by the seed to provision the cloud hosts. The `cloud` network carries all internal control plane and storage traffic, and is used by the control plane to provision the bare metal compute hosts. Finally, the `external` network links the cloud to the outside world.

We could describe such a network as follows:

Listing 1.1: networks.yml

```
---
# Network role mappings.
provision_oc_net_name: management
provision_wl_net_name: cloud
internal_net_name: cloud
external_net_name: external
storage_net_name: cloud
storage_mgmt_net_name: cloud
inspection_net_name: cloud

# management network definition.
management_cidr: 10.0.0.0/24
management_allocation_pool_start: 10.0.0.1
management_allocation_pool_end: 10.0.0.127
management_inspection_allocation_pool_start: 10.0.0.128
management_inspection_allocation_pool_end: 10.0.0.254

# cloud network definition.
cloud_cidr: 10.0.1.0/23
cloud_allocation_pool_start: 10.0.1.1
cloud_allocation_pool_end: 10.0.1.127
cloud_inspection_allocation_pool_start: 10.0.1.128
cloud_inspection_allocation_pool_end: 10.0.1.255
cloud_neutron_allocation_pool_start: 10.0.2.0
cloud_neutron_allocation_pool_end: 10.0.2.254

# external network definition.
external_cidr: 10.0.3.0/24
external_allocation_pool_start: 10.0.3.1
external_allocation_pool_end: 10.0.3.127
external_neutron_allocation_pool_start: 10.0.3.128
external_neutron_allocation_pool_end: 10.0.3.254
```

We can map these networks to network interfaces on the seed and controller hosts:

Listing 1.2: inventory/group_vars/seed/network-interfaces

```
---
management_interface: eth0
```

Listing 1.3: inventory/group_vars/controllers/network-interfaces

```
---
management_interface: eth0
cloud_interface: breth1
cloud_bridge_ports:
  - eth1
external_interface: eth2
```

We have defined a bridge for the cloud network on the controllers as this will allow it to be plugged into a neutron Open vSwitch bridge.

Kayobe will allocate IP addresses for the hosts that it manages:

Listing 1.4: network-allocation.yml

```
---
management_ips:
  seed: 10.0.0.1
  control0: 10.0.0.2
  control1: 10.0.0.3
  control2: 10.0.0.4
cloud_ips:
  control0: 10.0.1.1
  control1: 10.0.1.2
  control2: 10.0.1.3
external_ips:
  control0: 10.0.3.1
  control1: 10.0.3.2
  control2: 10.0.3.3
```

Note that although this file does not need to be created manually, doing so allows for a predictable IP address mapping which may be desirable in some cases.

Usage

This section describes usage of Kayobe to install an OpenStack cloud onto a set of bare metal servers. We assume access is available to a node which will act as the hypervisor hosting the seed node in a VM. We also assume that this seed hypervisor has access to the bare metal nodes that will form the OpenStack control plane. Finally, we assume that the control plane nodes have access to the bare metal nodes that will form the workload node pool.

Command Line Interface

Note: Where a prompt starts with `(kayobe-venv)` it is implied that the user has activated the Kayobe virtualenv. This can be done as follows:

```
$ source kayobe-venv/bin/activate
```

To deactivate the virtualenv:

```
(kayobe-venv) $ deactivate
```

To see information on how to use the `kayobe` CLI and the commands it provides:

```
(kayobe-venv) $ kayobe help
```

As the `kayobe` CLI is based on the `cliff` package (as used by the `openstack` client), it supports tab auto-completion of subcommands. This can be activated by generating and then sourcing the bash completion script:

```
(kayobe-venv) $ kayobe complete > kayobe-complete
(kayobe-venv) $ source kayobe-complete
```

Working with Ansible Vault

If Ansible vault has been used to encrypt Kayobe configuration files, it will be necessary to provide the `kayobe` command with access to vault password. There are three options for doing this:

Prompt Use `kayobe --ask-vault-pass` to prompt for the password.

File Use `kayobe --vault-password-file <file>` to read the password from a (plain text) file.

Environment variable Export the environment variable `KAYOBE_VAULT_PASSWORD` to read the password from the environment.

Ansible Control Host

Before starting deployment we must bootstrap the Ansible control host. Tasks performed here include:

- Install Ansible and role dependencies from Ansible Galaxy.
- Generate an SSH key if necessary and add it to the current user's authorised keys.
- Configure kolla and kolla-ansible.

To bootstrap the Ansible control host:

```
(kayobe-venv) $ kayobe control host bootstrap
```

Physical Network

The physical network can be managed by Kayobe, which uses Ansible's network modules. Currently Dell Network OS 6 and Dell Network OS 9 switches are supported but this could easily be extended. To provision the physical network:

```
(kayobe-venv) $ kayobe physical network configure --group <group> [--enable-discovery]
```

The `--group` argument is used to specify an Ansible group containing the switches to be configured.

The `--enable-discovery` argument enables a one-time configuration of ports attached to baremetal compute nodes to support hardware discovery via ironic inspector.

Seed

VM Provisioning

Note: It is not necesary to run the seed services in a VM. To use an existing bare metal host or a VM provisioned outside of Kayobe, this step may be skipped. Ensure that the Ansible inventory contains a host for the seed.

The seed hypervisor should have CentOS and `libvirt` installed. It should have `libvirt` networks configured for all networks that the seed VM needs access to and a `libvirt` storage pool available for the seed VM's volumes. To provision the seed VM:

```
(kayobe-venv) $ kayobe seed vm provision
```

When this command has completed the seed VM should be active and accessible via SSH. Kayobe will update the Ansible inventory with the IP address of the VM.

Host Configuration

To configure the seed host OS:

```
(kayobe-venv) $ kayobe seed host configure
```

Note: If the seed host uses disks that have been in use in a previous installation, it may be necessary to wipe partition and LVM data from those disks. To wipe all disks that are not mounted during host configuration:

```
(kayobe-venv) $ kayobe seed host configure --wipe-disks
```

Building Container Images

Note: It is possible to use prebuilt container images from an image registry such as Dockerhub. In this case, this step can be skipped.

It is possible to use prebuilt container images from an image registry such as Dockerhub. In some cases it may be necessary to build images locally either to apply local image customisation or to use a downstream version of kolla. To build images locally:

```
(kayobe-venv) $ kayobe seed container image build
```

It is possible to build a specific set of images by supplying one or more image name regular expressions:

```
(kayobe-venv) $ kayobe seed container image build bifrost-deploy
```

In order to push images to a registry after they are built, add the `--push` argument.

Deploying Containerised Services

At this point the seed services need to be deployed on the seed VM. These services are deployed in the `bifrost_deploy` container. This command will also build the Operating System image that will be used to deploy the overcloud nodes using Disk Image Builder (DIB).

To deploy the seed services in containers:

```
(kayobe-venv) $ kayobe seed service deploy
```

After this command has completed the seed services will be active.

Accessing the Seed via SSH (Optional)

For SSH access to the seed, first determine the seed's IP address. We can use the `kayobe configuration dump` command to inspect the seed's IP address:

```
(kayobe-venv) $ kayobe configuration dump --host seed --var-name ansible_host
```

The `kayobe_ansible_user` variable determines which user account will be used by Kayobe when accessing the machine via SSH. By default this is `stack`. Use this user to access the seed:

```
$ ssh <kayobe ansible user>@<seed VM IP>
```

To see the active Docker containers:

```
$ docker ps
```

Leave the seed VM and return to the shell on the control host:

```
$ exit
```

Overcloud

Discovery

Note: If discovery of the overcloud is not possible, a static inventory of servers using the bifrost `servers.yml` file format may be configured using the `kolla_bifrost_servers` variable in `${KAYOBE_CONFIG_PATH}/bifrost.yml`.

Discovery of the overcloud is supported by the ironic inspector service running in the `bifrost_deploy` container on the seed. The service is configured to PXE boot unrecognised MAC addresses with an IPA ramdisk for introspection. If an introspected node does not exist in the ironic inventory, ironic inspector will create a new entry for it.

Discovery of the overcloud is triggered by causing the nodes to PXE boot using a NIC attached to the overcloud provisioning network. For many servers this will be the factory default and can be performed by powering them on.

On completion of the discovery process, the overcloud nodes should be registered with the ironic service running in the seed host's `bifrost_deploy` container. The node inventory can be viewed by executing the following on the seed:

```
$ docker exec -it bifrost_deploy bash
(bifrost_deploy) $ source env-vars
(bifrost_deploy) $ ironic node-list
```

In order to interact with these nodes using Kayobe, run the following command to add them to the Kayobe and bifrost Ansible inventories:

```
(kayobe-venv) $ kayobe overcloud inventory discover
```

BIOS and RAID Configuration

Note: BIOS and RAID configuration may require one or more power cycles of the hardware to complete the operation. These will be performed automatically.

Configuration of BIOS settings and RAID volumes is currently performed out of band as a separate task from hardware provisioning. To configure the BIOS and RAID:

```
(kayobe-venv) $ kayobe overcloud bios raid configure
```

After configuring the nodes' RAID volumes it may be necessary to perform hardware inspection of the nodes to reconfigure the ironic nodes' scheduling properties and root device hints. To perform manual hardware inspection:

```
(kayobe-venv) $ kayobe overcloud hardware inspect
```

Provisioning

Provisioning of the overcloud is performed by the ironic service running in the bifrost container on the seed. To provision the overcloud nodes:

```
(kayobe-venv) $ kayobe overcloud provision
```

After this command has completed the overcloud nodes should have been provisioned with an OS image. The command will wait for the nodes to become `active` in ironic and accessible via SSH.

Host Configuration

To configure the overcloud hosts' OS:

```
(kayobe-venv) $ kayobe overcloud host configure
```

Note: If the controller hosts use disks that have been in use in a previous installation, it may be necessary to wipe partition and LVM data from those disks. To wipe all disks that are not mounted during host configuration:

```
(kayobe-venv) $ kayobe overcloud host configure --wipe-disks
```

Building Container Images

Note: It is possible to use prebuilt container images from an image registry such as Dockerhub. In this case, this step can be skipped.

In some cases it may be necessary to build images locally either to apply local image customisation or to use a downstream version of kolla. To build images locally:

```
(kayobe-venv) $ kayobe overcloud container image build
```

It is possible to build a specific set of images by supplying one or more image name regular expressions:

```
(kayobe-venv) $ kayobe overcloud container image build ironic- nova-api
```

In order to push images to a registry after they are built, add the `--push` argument.

Pulling Container Images

Note: It is possible to build container images locally avoiding the need for an image registry such as Dockerhub. In this case, this step can be skipped.

In most cases suitable prebuilt kolla images will be available on Dockerhub. The stackhpc account provides image repositories suitable for use with kayobe and will be used by default. To pull images from the configured image registry:

```
(kayobe-venv) $ kayobe overcloud container image pull
```

Deploying Containerised Services

To deploy the overcloud services in containers:

```
(kayobe-venv) $ kayobe overcloud service deploy
```

Once this command has completed the overcloud nodes should have OpenStack services running in Docker containers.

Interacting with the Control Plane

Kolla-ansible writes out an environment file that can be used to access the OpenStack admin endpoints as the admin user:

```
$ source ${KOLLA_CONFIG_PATH:-/etc/kolla}/admin-openrc.sh
```

Kayobe also generates an environment file that can be used to access the OpenStack public endpoints as the admin user which may be required if the admin endpoints are not available from the control host:

```
$ source ${KOLLA_CONFIG_PATH:-/etc/kolla}/public-openrc.sh
```

Performing Post-deployment Configuration

To perform post deployment configuration of the overcloud services:

```
(kayobe-venv) $ source ${KOLLA_CONFIG_PATH:-/etc/kolla}/admin-openrc.sh
(kayobe-venv) $ kayobe overcloud post configure
```

This will perform the following tasks:

- Register Ironic Python Agent (IPA) images with glance
- Register introspection rules with ironic inspector
- Register a provisioning network and subnet with neutron

Reconfiguring Containerised Services

When configuration is changed, it is necessary to apply these changes across the system in an automated manner. To reconfigure the overcloud, first make any changes required to the configuration on the control host. Next, run the following command:

```
(kayobe-venv) $ kayobe overcloud service reconfigure
```

In case not all services' configuration have been modified, performance can be improved by specifying Ansible tags to limit the tasks run in kayobe and/or kolla-ansible's playbooks. This may require knowledge of the inner workings of these tools but in general, kolla-ansible tags the play used to configure each service by the name of that service. For

example: `nova`, `neutron` or `ironic`. Use `-t` or `--tags` to specify kayobe tags and `-kt` or `--kolla-tags` to specify kolla-ansible tags. For example:

```
(kayobe-venv) $ kayobe overcloud service reconfigure --tags config --kolla-tags nova,
↪ironic
```

Upgrading Containerised Services

Containerised control plane services may be upgraded by replacing existing containers with new containers using updated images which have been pulled from a registry or built locally. If using an updated version of Kayobe, it should be upgraded before upgrading the overcloud services. If Kayobe has been upgraded, ensure that any required configuration changes have been performed as described in the release notes.

To upgrade the containerised control plane services:

```
(kayobe-venv) $ kayobe overcloud service upgrade
```

As for the reconfiguration command, it is possible to specify tags for Kayobe and/or kolla-ansible:

```
(kayobe-venv) $ kayobe overcloud service upgrade --tags config --kolla-tags keystone
```

Other Useful Commands

Deprovisioning

Note: This step will power down the overcloud hosts and delete their nodes' instance state from the seed's ironic service.

To deprovision the overcloud:

```
(kayobe-venv) $ kayobe overcloud deprovision
```

Running Kayobe Playbooks on Demand

In some situations it may be necessary to run an individual Kayobe playbook. Playbooks are stored in `<kayobe repo>/ansible/*.yml`. To run an arbitrary Kayobe playbook:

```
(kayobe-venv) $ kayobe playbook run <playbook> [<playbook>]
```

Running Kolla-ansible Commands

To execute a kolla-ansible command:

```
(kayobe-venv) $ kayobe kolla ansible run <command>
```

Dumping Kayobe Configuration

The Ansible configuration space is quite large, and it can be hard to determine the final values of Ansible variables. We can use Kayobe's `configuration dump` command to view individual variables or the variables for one or more hosts. To dump Kayobe configuration for one or more hosts:

```
(kayobe-venv) $ kayobe configuration dump
```

The output is a JSON-formatted object mapping hosts to their hostvars.

We can use the `--var-name` argument to inspect a particular variable or the `--host` or `--hosts` arguments to view a variable or variables for a specific host or set of hosts.

Developer Documentation

How to Contribute

Kayobe does not currently follow the upstream OpenStack development process, but we will still be incredibly grateful for any contributions.

Please raise issues and submit pull requests via Github.

Thanks in advance!