kayobe Documentation

Release

OpenStack Foundation

Sep 20, 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://kayobe.readthedocs.io/en/latest/
- 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Note: Kayobe and its documentation is currently under heavy development, and therefore may be incomplete or out of date.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, network, monitoring, 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.

Cloud Hosts

Cloud hosts can further be divided into subclasses.

Controllers Controller hosts run the OpenStack control plane services.

Network Network hosts run the neutron networking services and load balancers for the OpenStack API services.

Monitoring Monitoring host run the control plane and workload monitoring services. Currently, kayobe does not deploy any services onto monitoring hosts.

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 out-of-band network Name of the network used by the seed to access the out-of-band management controllers of the bare metal overcloud hosts.

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

Workload out-of-band network Name of the network used by the overcloud hosts to access the out-of-band management controllers of the bare metal workload 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.

Public network The public network hosts the public OpenStack API endpoints.

External network The external network provides external network access for the hosts in the system.

Installation

Prerequisites

Currently Kayobe supports the following Operating Systems on the control host:

- CentOS 7.3
- Ubuntu 16.04

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. It is necessary to install the GCC compiler chain in order to build the extensions of some of kayobe's python dependencies. Finally, for cloning and working with the kayobe source code repository, Git is required.

On CentOS:

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

On Ubuntu:

```
$ apt install -y python-virtualenv gcc git
```

Installation

This guide will describe how to install Kayobe from source in a virtualenv.

The directory structure for a kayobe control host environment is configurable, but the following is recommended, where `<base_path>` is the path to a top level directory:

```
<base_path>/
    src/
        kayobe/
        kayobe-config/
        kolla-ansible/
    venvs/
        kayobe/
        kolla-ansible/
```

First, change to the top level directory, and make the directories for source code repositories and python virtual environments:

```
$ cd <base_path>
$ mkdir -p src venvs
```

Next, obtain the Kayobe source code. For example:

```
$ cd <base_path>/src
$ git clone https://github.com/stackhpc/kayobe
```

Create a virtualenv for Kayobe:

```
$ virtualenv <base_path>/venvs/kayobe
```

Activate the virtualenv and update pip:

```
$ source <base_path>/venvs/kayobe/bin/activate
(kayobe) $ pip install -U pip
```

Install Kayobe and its dependencies using the source code checkout:

```
(kayobe) $ cd <base_path>/src/kayobe
(kayobe) $ pip install .
```

Finally, deactivate the virtualenv:

```
(kayobe) $ deactivate
```

Creation of a `kayobe-config` source code repository will be covered in the *configuration guide*. The kolla-ansible source code checkout and python virtual environment will be created automatically by kayobe.

Usage

Command Line Interface

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

```
$ source kayobe/bin/activate
```

To deactivate the virtualenv:

```
(kayobe) $ deactivate
```

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

```
(kayobe) $ 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) $ kayobe complete > kayobe-complete
(kayobe) $ 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.

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 if discovery of the control plane hardware is planned, although entries for groups may still be required.

Use of advanced control planes with multiple server roles and customised service placement across those servers is covered in *Control Plane Service Placement*.

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.

inspection_gateway IP address of the gateway for the hardware introspection network.

neutron_gateway IP address of the gateway for a neutron subnet based on this network.

vlan VLAN ID.

mtu Maximum Transmission Unit (MTU).

routes List of static IP routes. Each item should be a dict containing the items `cidr` and `gateway`. `cidr` is the CIDR representation of the route's destination. `gateway` is the IP address of the next hop.

physical_network Name of the physical network on which this network exists. This aligns with the physical network concept in neutron.

libvirt_network_name A name to give to a Libvirt network representing this network on the seed hypervisor.

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.

bond_mode For bond interfaces, the bond's mode, e.g. 802.3ad.

bond_slaves For bond interfaces, a list of names of network interfaces to act as slaves for the bond.

bond_miimon For bond interfaces, the time in milliseconds between MII link monitoring.

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:

oob_oc_net_name Name of the network used by the seed to access the out-of-band management controllers of the bare metal overcloud hosts.

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

oob_wl_net_name Name of the network used by the overcloud hosts to access the out-of-band management controllers of the bare metal workload 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.

public_net_name Name of the network used to expose the public OpenStack API endpoints.

external_net_name Name of the network used 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.

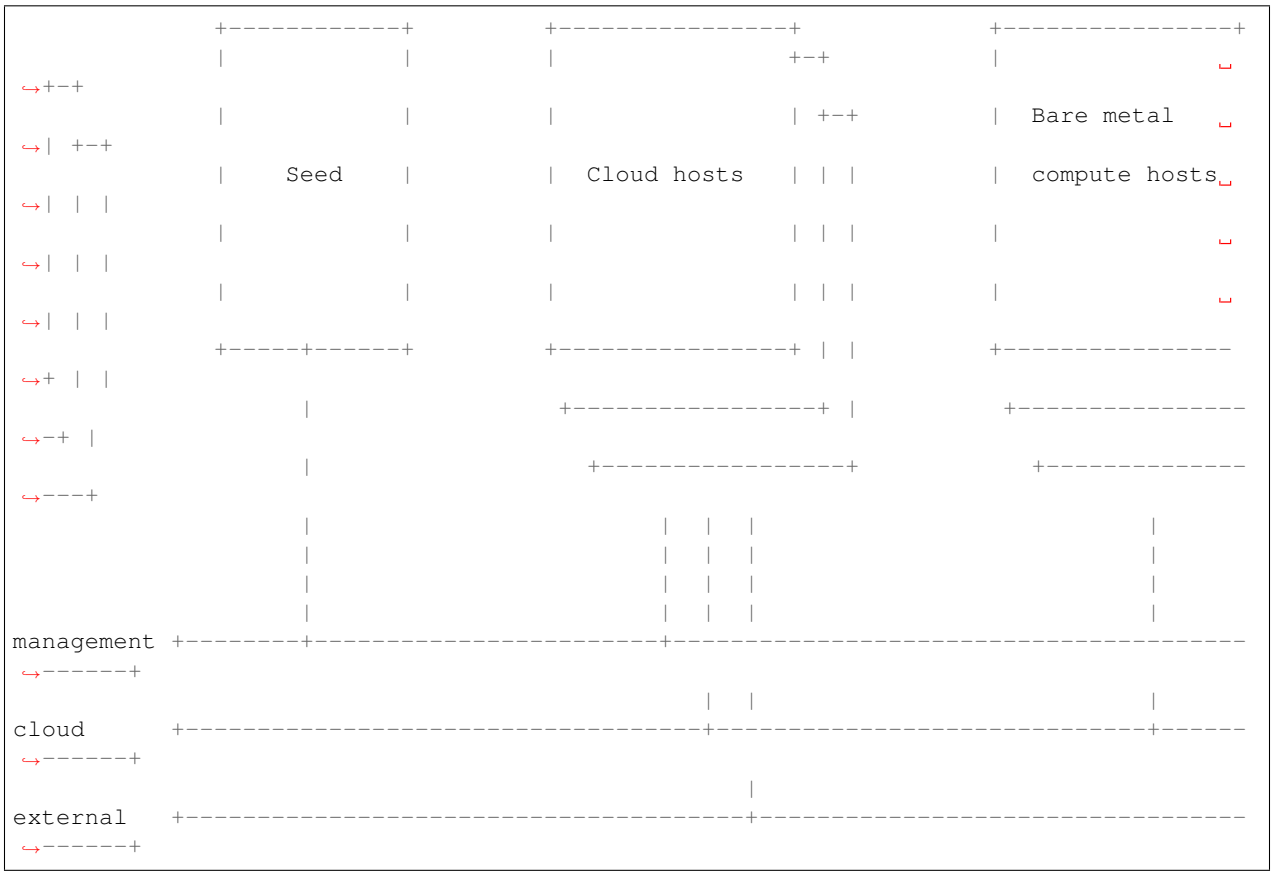
inspection_net_name Name of the network used to perform hardware introspection on the bare metal workload hosts.

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
public_net_name: external
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
external_routes:
  - cidr 10.0.4.0/24
    gateway: 10.0.3.1
```

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.

Kolla-ansible Configuration

Kayobe relies heavily on kolla-ansible for deployment of the OpenStack control plane. Kolla-ansible is installed locally on the ansible control host (the host from which kayobe commands are executed), and kolla-ansible commands are executed from there.

Local Environment

Environment variables are used to configure the environment in which kolla-ansible is installed and executed.

Table 1.1: Kolla-ansible environment variables

Variable	Purpose	Default
$KOLLA_CONFIG_PATH	Path on the ansible control host in which the kolla-ansible configuration will be generated.	/etc/kolla
$KOLLA_SOURCE_PATH	Path on the ansible control host in which the kolla-ansible source code will be cloned.	$PWD/src/kolla-ansible
$KOLLA_VENV_PATH	Path on the ansible control host in which the kolla-ansible virtualenv will be created.	$PWD/venvs/kolla-ansible

Deployment

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.

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.

To bootstrap the Ansible control host:

```
(kayobe) $ 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) $ 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 Hypervisor

Note: It is not necessary to run the seed services in a VM. To use an existing bare metal host or a VM provisioned outside of Kayobe, this section may be skipped.

Host Configuration

To configure the seed hypervisor's host OS, and the Libvirt/KVM virtualisation support:

```
(kayobe) $ kayobe seed hypervisor host configure
```

Seed

VM Provisioning

Note: It is not necessary 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) $ 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) $ 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) $ 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) $ 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) $ 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) $ kayobe seed service deploy
```

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

Building Deployment Images

Note: It is possible to use prebuilt deployment images. In this case, this step can be skipped.

It is possible to use prebuilt deployment images from the OpenStack hosted tarballs or another source. In some cases it may be necessary to build images locally either to apply local image customisation or to use a downstream version of

Ironic Python Agent (IPA). In order to build IPA images, the `ipa_build_images` variable should be set to `True`. To build images locally:

```
(kayobe) $ kayobe seed deployment image build
```

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) $ 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) $ kayobe overcloud inventory discover
```

Saving Hardware Introspection Data

If ironic inspector is in use on the seed host, introspection data will be stored in the local nginx service. This data may be saved to the control host:

```
(kayobe) $ kayobe overcloud introspection data save
```

`--output-dir` may be used to specify the directory in which introspection data files will be saved. `--output-format` may be used to set the format of the files.

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) $ 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) $ 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) $ 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) $ 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) $ 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) $ 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) $ 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) $ kayobe overcloud container image pull
```

Building Deployment Images

Note: It is possible to use prebuilt deployment images. In this case, this step can be skipped.

It is possible to use prebuilt deployment images from the OpenStack hosted tarballs or another source. In some cases it may be necessary to build images locally either to apply local image customisation or to use a downstream version of Ironic Python Agent (IPA). In order to build IPA images, the `ipa_build_images` variable should be set to `True`. To build images locally:

```
(kayobe) $ kayobe overcloud deployment image build
```

Deploying Containerised Services

To deploy the overcloud services in containers:

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

Upgrading

This section describes how to upgrade from one OpenStack release to another.

Preparation

Before you start, be sure to back up any local changes, configuration, and data.

Upgrading Kayobe

If a new, suitable version of kayobe is available, it should be installed. If using kayobe from a git checkout, this may be done by pulling down the new version from Github. Make sure that any local changes to kayobe are committed. For example, to pull version 1.0.0 from the `origin` remote:

```
$ git pull origin 1.0.0
```

If local changes were made to kayobe, these should now be reapplied.

The upgraded kayobe python module and dependencies should be installed:

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

Migrating Kayobe Configuration

Kayobe configuration options may be changed between releases of kayobe. Ensure that all site local configuration is migrated to the target version format. If using the kayobe-config git repository to manage local configuration, this process can be managed via git. For example, to fetch version 1.0.0 of the configuration from the `origin` remote and merge it into the current branch:

```
$ git fetch origin 1.0.0
$ git merge 1.0.0
```

The configuration should be manually inspected after the merge to ensure that it is correct. Any new configuration options may be set at this point. In particular, the following options may need to be changed if not using their default values:

- `kolla_openstack_release`
- `kolla_sources`
- `kolla_build_blocks`
- `kolla_build_customizations`

Once the configuration has been migrated, it is possible to view the global variables for all hosts:

```
(kayobe) $ kayobe configuration dump
```

The output of this command is a JSON object mapping hosts to their configuration. The output of the command may be restricted using the `--host`, `--hosts`, `--var-name` and `--dump-facts` options.

Upgrading the Control Host

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

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

To upgrade the Ansible control host:

```
(kayobe) $ kayobe control host upgrade
```

Upgrading the Seed

Currently, upgrading the seed services is not supported.

Upgrading the Overcloud

The overcloud services are upgraded in two steps. First, new container images should be obtained either by building them locally or pulling them from an image registry. Second, the overcloud services should be replaced with new containers created from the new container images.

Upgrading Host Services

Prior to upgrading the OpenStack control plane, the overcloud host services should be upgraded:

```
(kayobe) $ kayobe overcloud host upgrade
```

Note that this will not perform full configuration of the host, and will instead perform a targeted upgrade of specific services where necessary.

Upgrading the Ironic Deployment Images

Prior to upgrading the OpenStack control plane, the baremetal compute nodes should be configured to use an updated deployment ramdisk. This procedure is not currently automated by kayobe.

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) $ 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) $ 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) $ kayobe overcloud container image pull
```

Saving Overcloud Service Configuration

It is often useful to be able to save the configuration of the control plane services for inspection or comparison with another configuration set prior to a reconfiguration or upgrade. This command will gather and save the control plane configuration for all hosts to the ansible control host:

```
(kayobe) $ kayobe overcloud service configuration save
```

The default location for the saved configuration is `$PWD/overcloud-config`, but this can be changed via the `output-dir` argument. To gather configuration from a directory other than the default `/etc/kolla`, use the `node-config-dir` argument.

Generating Overcloud Service Configuration

Prior to deploying, reconfiguring, or upgrading a control plane, it may be useful to generate the configuration that will be applied, without actually applying it to the running containers. The configuration should typically be generated in a directory other than the default configuration directory of `/etc/kolla`, to avoid overwriting the active configuration:

```
(kayobe) $ kayobe overcloud service configuration generate --node-config-dir /path/to/
↪generated/config
```

The configuration will be generated remotely on the overcloud hosts in the specified directory, with one subdirectory per container. This command may be followed by `kayobe ovecloud service configuration save` to gather the generated configuration to the ansible control host.

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.

To upgrade the containerised control plane services:

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

It is possible to specify tags for Kayobe and/or kolla-ansible to restrict the scope of the upgrade:

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

Administration

This section describes how to use kayobe to simplify post-deployment administrative tasks.

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) $ 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) $ 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 or upgrading from one release of OpenStack to another, be sure to follow the *kayobe upgrade guide*. It may be necessary to upgrade one or more services within a release, for example to apply a patch or minor release.

To upgrade the containerised control plane services:

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

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

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

Destroying the Overcloud Services

Note: This step will destroy all containers, container images, volumes and data on the overcloud hosts.

To destroy the overcloud services:

```
(kayobe) $ kayobe overcloud service destroy --yes-i-really-really-mean-it
```

Deprovisioning The Cloud

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) $ kayobe overcloud deprovision
```

Deprovisioning The Seed VM

Note: This step will destroy the seed VM and its data volumes.

To deprovision the seed VM:

```
(kayobe) $ kayobe seed vm deprovision
```

Saving Overcloud Service Configuration

It is often useful to be able to save the configuration of the control plane services for inspection or comparison with another configuration set prior to a reconfiguration or upgrade. This command will gather and save the control plane configuration for all hosts to the ansible control host:

```
(kayobe) $ kayobe overcloud service configuration save
```

The default location for the saved configuration is `$PWD/overcloud-config`, but this can be changed via the `output-dir` argument. To gather configuration from a directory other than the default `/etc/kolla`, use the `node-config-dir` argument.

Generating Overcloud Service Configuration

Prior to deploying, reconfiguring, or upgrading a control plane, it may be useful to generate the configuration that will be applied, without actually applying it to the running containers. The configuration should typically be generated in a directory other than the default configuration directory of `/etc/kolla`, to avoid overwriting the active configuration:

```
(kayobe) $ kayobe overcloud service configuration generate --node-config-dir /path/to/
↪generated/config
```

The configuration will be generated remotely on the overcloud hosts in the specified directory, with one subdirectory per container. This command may be followed by `kayobe ovecloud service configuration save` to gather the generated configuration to the ansible control host.

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) $ kayobe playbook run <playbook> [<playbook>]
```

Running Kolla-ansible Commands

To execute a kolla-ansible command:

```
(kayobe) $ 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) $ 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.

Advanced Documentation

Control Plane Service Placement

Note: This is an advanced topic and should only be attempted when familiar with kayobe and OpenStack.

The default configuration in kayobe places all control plane services on a single set of servers described as 'controllers'. In some cases it may be necessary to introduce more than one server role into the control plane, and control which services are placed onto the different server roles.

Configuration

Overcloud Inventory Discovery

If using a seed host to enable discovery of the control plane services, it is necessary to configure how the discovered hosts map into kayobe groups. This is done using the `overcloud_group_hosts_map` variable, which maps names of kayobe groups to a list of the hosts to be added to that group.

This variable will be used during the command `kayobe overcloud inventory discover`. An inventory file will be generated in `${KAYOBE_CONFIG_PATH}/inventory/overcloud` with discovered hosts added to appropriate kayobe groups based on `overcloud_group_hosts_map`.

Kolla-ansible Inventory Mapping

Once hosts have been discovered and enrolled into the kayobe inventory, they must be added to the kolla-ansible inventory. This is done by mapping from top level kayobe groups to top level kolla-ansible groups using the `kolla_overcloud_inventory_top_level_group_map` variable. This variable maps from kolla-ansible groups to lists of kayobe groups, and variables to define for those groups in the kolla-ansible inventory.

Variables For Custom Server Roles

Certain variables must be defined for hosts in the `overcloud` group. For hosts in the `controllers` group, many variables are mapped to other variables with a `controller_` prefix in files under `ansible/group_vars/controllers/`. This is done in order that they may be set in a global extra variables file, typically `controllers.yml`, with defaults set in `ansible/group_vars/all/controllers`. A similar scheme is used for hosts in the `monitoring` group.

Table 1.2: Overcloud host variables

Variable	Purpose
`ansible_user`	Username with which to access the host via SSH.
`bootstrap_user`	Username with which to access the host before `ansible_user` is configured.
`lvm_groups`	List of LVM volume groups to configure. See mrlesmithjr.manage-lvm role for format.
`network_interfaces`	List of names of networks to which the host is connected.
`sysctl_parameters`	Dict of sysctl parameters to set.
`users`	List of users to create. See singleplatform-eng.users role

If configuring BIOS and RAID via `kayobe overcloud bios raid configure`, the following variables should also be defined:

Table 1.3: Overcloud BIOS & RAID host variables

Variable	Purpose
`bios_config`	Dict mapping BIOS configuration options to their required values. See stackhpc.drac role for format.
`raid_config`	List of RAID virtual disks to configure. See stackhpc.drac role for format.

These variables can be defined in inventory host or group variables files, under `${KAYOBE_CONFIG_PATH}/inventory/host_vars/<host>` or `${KAYOBE_CONFIG_PATH}/inventory/group_vars/<group>` respectively.

Custom Kolla-ansible Inventories

As an advanced option, it is possible to fully customise the content of the kolla-ansible inventory, at various levels. To facilitate this, kayobe breaks the kolla-ansible inventory into three separate sections.

Top level groups define the roles of hosts, e.g. `controller` or `compute`, and it is to these groups that hosts are mapped directly.

Components define groups of services, e.g. `nova` or `ironic`, which are mapped to top level groups.

Services define single containers, e.g. `nova-compute` or `ironic-api`, which are mapped to components.

The default top level inventory is generated from `kolla_overcloud_inventory_top_level_group_map`. Kayobe's component- and service-level inventory for kolla-ansible is static, and taken from the kolla-ansible example `multinode` inventory. The complete inventory is generated by concatenating these inventories.

Each level may be separately overridden by setting the following variables:

Table 1.4: Custom kolla-ansible inventory variables

Variable	Purpose
`kolla_overcloud_inventory_custom_top_level`	Overcloud inventory containing a mapping from top level groups to hosts.
`kolla_overcloud_inventory_custom_components`	Overcloud inventory containing a mapping from components to top level groups.
`kolla_overcloud_inventory_custom_services`	Overcloud inventory containing a mapping from services to components.
`kolla_overcloud_inventory_custom`	Full overcloud inventory contents.

Examples

Example 1: Adding Network Hosts

This example walks through the configuration that could be applied to enable the use of separate hosts for neutron network services and load balancing. The control plane consists of three controllers, `controller-[0-2]`, and two network hosts, `network-[0-1]`. All file paths are relative to `${KAYOBE_CONFIG_PATH}`.

First, we must map the hosts to kayobe groups.

Listing 1.5: `overcloud.yml`

```
overcloud_group_hosts_map:
  controllers:
    - controller-0
    - controller-1
    - controller-2
  network:
    - network-0
    - network-1
```

Next, we must map these groups to kolla-ansible groups.

Listing 1.6: `kolla.yml`

```
kolla_overcloud_inventory_top_level_group_map:
  control:
    groups:
      - controllers
  network:
    groups:
      - network
```

Finally, we create a group variables file for hosts in the network group, providing the necessary variables for a control plane host.

Listing 1.7: `inventory/group_vars/network`

```
ansible_user: "{{ kayobe_ansible_user }}"
bootstrap_user: "{{ controller_bootstrap_user }}"
lvm_groups: "{{ controller_lvm_groups }}"
network_interfaces: "{{ controller_network_host_network_interfaces }}"
sysctl_parameters: "{{ controller_sysctl_parameters }}"
users: "{{ controller_users }}"
```

Here we are using the controller-specific values for some of these variables, but they could equally be different.

Example 2: Overriding the Kolla-ansible Inventory

This example shows how to override one or more sections of the kolla-ansible inventory. All file paths are relative to `${KAYOBE_CONFIG_PATH}`.

First, create a file containing the customised inventory section. We'll use the **components** section in this example.

Listing 1.8: `kolla/inventory/overcloud-components.j2`

```
[nova]
control

[ironic]
{% if kolla_enable_ironic | bool %}
control
{% endif %}

...
```

Next, we must configure kayobe to use this inventory template.

Listing 1.9: `kolla.yml`

```
kolla_overcloud_inventory_custom_components: "{{ lookup('template', kayobe_config_
↪path ~ '/kolla/inventory/overcloud-components.j2') }}"
```

Here we use the `template` lookup plugin to render the Jinja2-formatted inventory template.

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!

Development

This section describes how to set up an OpenStack controller in a virtual machine using Vagrant and Kayobe.

Preparation

First, ensure that Vagrant is installed and correctly configured to use virtual box. Also install the following vagrant plugins:

vagrant plugin install vagrant-vbguest vagrant plugin install vagrant-reload

Note: if using Ubuntu 16.04 LTS, you may be unable to install any plugins. To work around this install the upstream version from www.virtualbox.org.

Next, clone kayobe:

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

Change the current directory to the kayobe repository:

```
cd kayobe
```

Inspect kayobe's `Vagrantfile`, noting the provisioning steps:

```
less Vagrantfile
```

Bring up a virtual machine:

```
vagrant up
```

Wait for the VM to boot.

Installation

SSH into the controller VM:

```
vagrant ssh
```

Source the kayobe virtualenv activation script:

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

Change the current directory to the Vagrant shared directory:

```
cd /vagrant
```

Source the kayobe environment file:

```
source kayobe-env
```

Bootstrap the kayobe control host:

```
kayobe control host bootstrap
```

Configure the controller host:

```
kayobe overcloud host configure
```

During execution of this command, SELinux will be disabled and the VM will be rebooted, causing you to be logged out. Wait for the VM to finish rebooting and log in, performing the same environment setup steps as before:

```
vagrant ssh
source kayobe-venv/bin/activate
cd /vagrant
source kayobe-env
```

Run the host configuration command again to completion:

```
kayobe overcloud host configure
```

At this point, container images must be acquired. They can either be built locally or pulled from an image repository if appropriate images are available.

Either build container images:

```
kayobe overcloud container image build
```

Or pull container images:

```
kayobe overcloud container image pull
```

Deploy the control plane services:

```
kayobe overcloud service deploy
```

Source the OpenStack environment file:

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

Perform post-deployment configuration:

```
kayobe overcloud post configure
```

Next Steps

The OpenStack control plane should now be active. Try out the following:

- register a user
- create an image
- upload an SSH keypair
- access the horizon dashboard

The cloud is your oyster!

To Do

Create virtual baremetal nodes to be managed by the OpenStack control plane.

Release Notes

Release Notes

Kayobe 2.0.0

Kayobe 2.0.0 was released on 15th September 2017.

Features

- Adds support for configuration of networks for out-of-band management for the overcloud and control plane hosts via the `oob_oc_net_name` and `oob_wl_net_name` variables respectively.
- Adds support for configuration of a *seed hypervisor* host. This host runs the *seed VM*. Currently, configuration of host networking, NTP, and libvirt storage pools and networks is supported.
- Adds a `base_path` variable to simplify configuration of paths. This is used to set the default value of `image_cache_path` and `source_checkout_path`. The default value of the base path may be set by the `$KAYOBE_BASE_PATH` environment variable.
- Adds a `virtualenv_path` variable to configure the path on which to create virtual environments.
- Uses the CentOS 7 cloud image for the seed VM by default.
- Adds a command to deprovision the seed VM, `kayobe seed vm deprovision`.
- Adds support for configuration of Juniper switches.
- Adds support for bonded (LAG) host network interfaces.
- Adds support for the overlay docker storage driver on the seed and overcloud hosts.
- Improves the Vagrant development environment, and provides configuration for a single controller with a single network.

- Adds support for building customised Ironic Python Agent (IPA) deployment images using Diskimage Builder (DIB). These can be built using the commands `kayobe seed deployment image build` and `kayobe overcloud deployment image build`.
- Adds a command to save overcloud introspection data, `kayobe overcloud introspection data save`.
- Separates the external network into external and public networks. The public network carries public API traffic, and is configured via `public_net_name`.
- Adds a `network` group, with networking and load balancing services moved to it. The group is a subgroup of the `controllers` group by default.
- Decomposes the overcloud inventory into top level, components, and services. This allows a deployer to customise their inventory at various levels, by providing a custom inventory template for one or more sections of the inventory.
- Adds support for configuration of sysctl parameters on the seed, seed hypervisor and overcloud hosts.
- Adds an **inspection-store** container for storage of workload hardware inspection data in environments without Swift.
- Adds configuration of gatewys in provisioning and inspection networks.
- Adds support for free-form configuration of Glance.
- Adds support for Ubuntu control hosts.
- Adds support for passing through host variables from kayobe to kolla-ansible. By default `ansible_host`, `ansible_port`, and `ansible_ssh_private_key_file`.

Upgrade Notes

- It is no longer necessary to set the `seed_vm_interfaces` variable, as the seed VM's network interfaces are now determined by the standard `seed_network_interfaces` variable.
- If using a CentOS 7 cloud image for the seed VM, it is no longer necessary to set the `seed_vm_root_image` variable.
- The default value of `kolla_enable_haproxy` has been changed to `True`.
- If using a custom inventory, a `network` group should be added to it. If the control hosts are providing networking services, then the `network` group should be a subgroup of the `controllers` group.
- The `overcloud_groups` variable is now determined more intelligently, and it is generally no longer necessary to set it manually.
- The provisioning network is now used to access the TFTP server during workload hardware inspection.
- A default gateway may be advertised to compute nodes during workload inspection, allowing access to an ironic inspector API on the internal API network.

Kayobe 1.1.0

Kayobe 1.1.0 was released on 17th July 2017.

Features

- Support static routes on control plane networks

- Improve documentation
- Initial support for in-development Pike release
- Upgrade kayobe control host & control plane
- Support overcloud service destroy command
- Support fluentd custom output configuration

Kayobe 1.0.0

1.0.0 is the first 'official' release of the Kayobe OpenStack deployment tool. It was released on 29th June 2017.

Features

This release includes the following 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