Reference Architectures 2017

Automate Red Hat OpenShift Container Platform Deployment on HPE ProLiant Servers with Ansible Tower and HPE OneView

David Critch Ken Bell

Reference Architectures 2017 Automate Red Hat OpenShift Container Platform Deployment on HPE ProLiant Servers with Ansible Tower and HPE OneView

David Critch

Ken Bell
refarch-feedback@redhat.com

Legal Notice

Abstract

The purpose of this document is to provide guidelines and considerations for installing and configuring Red Hat OpenShift Platform on HPE DL ProLiant Servers.

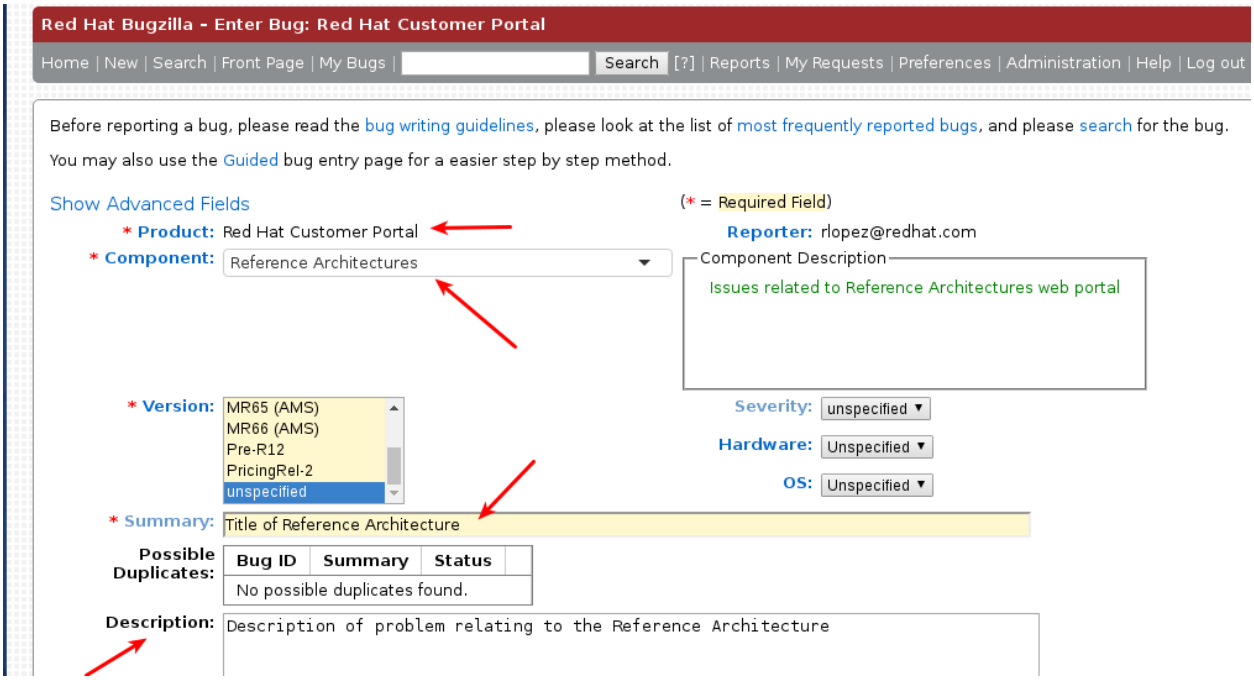
Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com or creating a bug report at bugzilla.redhat.com. Please refer to the title of the reference architecture within the email or bug report.

Providing feedback using a bug report allows both the person reporting the bug and us to track the feedback.

When filing a bug report for a reference architecture, create the bug under the Red Hat Customer Portal product and select the Component labeled Reference Architectures. Within the Summary, enter the title of the reference architecture. Within the Description, provide a URL (if available) along with any feedback.



Note

The scripts described in this reference architecture and provided in the associated git hub repo are not supported by Red Hat. They merely provide a mechanism to build out your infrastructure.

CHAPTER 1. EXECUTIVE SUMMARY

Container technology is rapidly changing the way organizations develop, deploy, and manage applications. Red Hat OpenShift Container Platform provides organizations with a reliable platform for deploying and scaling container based applications. This reference architecture will provide an example of how to deploy Red Hat OpenShift Container Platform on HPE ProLiant DL baremetal servers using Ansible Tower, Red Hat Satellite, and HPE OneView.

Intended Audience

This document is intended for systems engineers, systems administrators, architects, and installers responsible for installing and maintaining Red Hat OpenShift Container Platform. The reader of this document should be familiar with Red Hat OpenShift Container Platform, Red Hat Satellite, Ansible Tower, Red Hat Enterprise Linux®, HPE ProLiant servers, and networking equipment.

CHAPTER 2. SOLUTION DESIGN

2.1. OVERVIEW

This solution was designed to provide guidance for automating the deployment of Red Hat OpenShift Container Platform 3.5 on HPE ProLiant DL rack mounted servers. Deployment automation is conducted leveraging Ansible Tower playbooks and workflows. The Ansible playbooks are used to integrate with HPE OneView, Red Hat Satellite, Red Hat OpenShift Container Platform, and Red Hat Container-native storage. Ansible workflows coordinate the execution of the Ansible playbooks allowing a one-click deployment of the entire OpenShift Container Platform environment on nine ProLiant DL baremetal servers.

This solution design deploys Red Hat OpenShift Container Platform directly on baremetal physical servers. This is slightly different than other private and public cloud based deployment models where Red Hat OpenShift Container Platform is deployed on virtual machine instances. Advantages of deploying Red Hat OpenShift on physical servers is the reduction in the amount of operating systems which results in a smaller attack surface, reduced management, and reduced licensing that are typically associated with a hypervisor based solution. In addition, some performance benefits may be realized without the additional virtualization layer and associated overhead of running virtual machines.

Below is a high level description of the solution workflow. Detailed information on the Ansible Tower, HPE OneView, Red Hat Satellite, and Red Hat OpenShift Container Platform will be described in subsequent chapters.

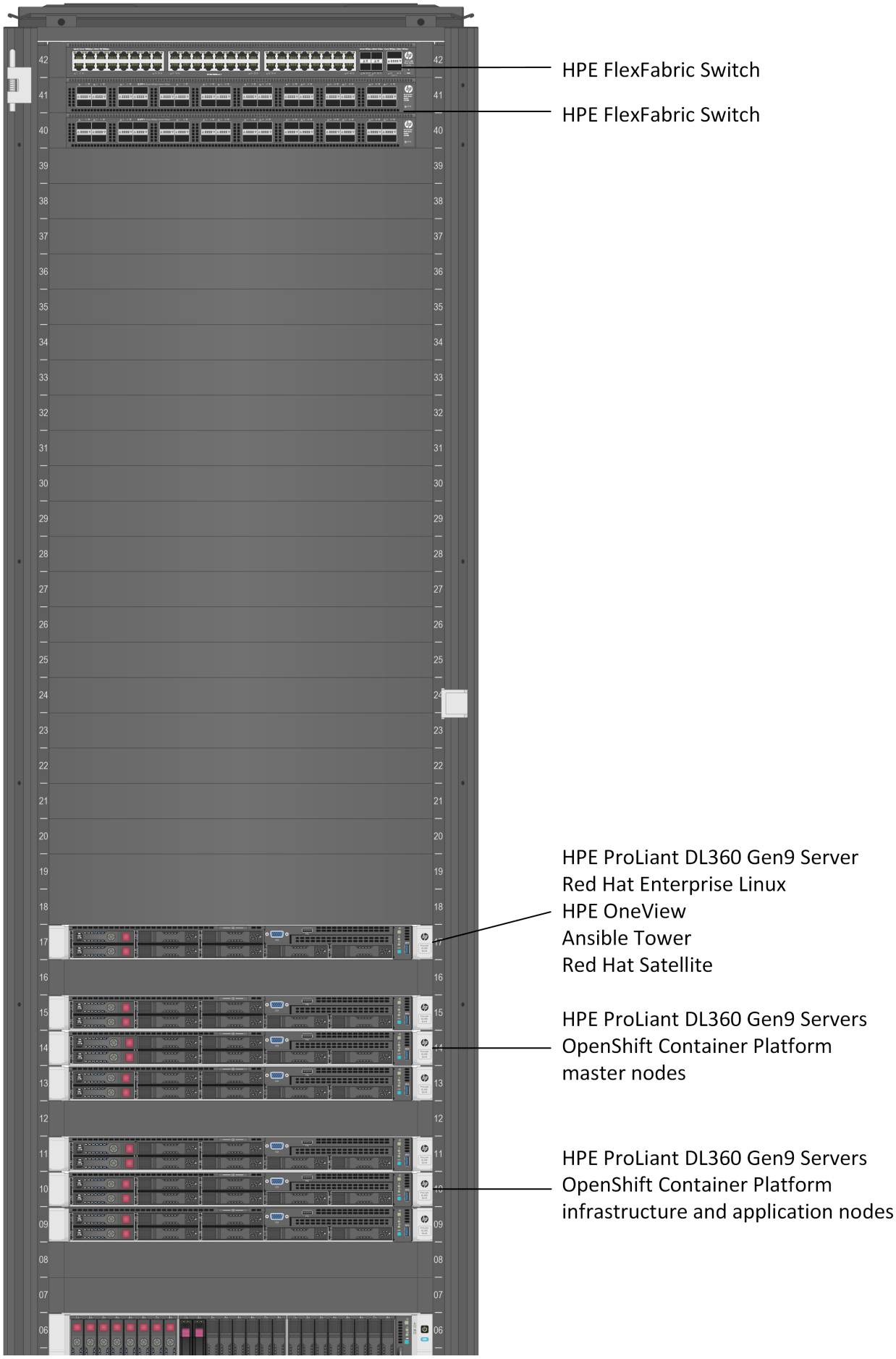
Red Hat Ansible Tower will leverage the HPE Ansible modules for OneView to register the physical servers with HPE OneView 3.1 and create server profiles for each of their respective server roles. The server profiles will be applied to the servers and the latest firmware baseline will be installed. Local disk configurations will be applied to the smart array controllers: RAID 1 for the operating systems and RAID 6 for the Container-native storage nodes. Ansible Tower playbooks will register the servers with Red Hat Satellite and Red Hat Enterprise Linux 7.3 will be installed on each of the ProLiant DL servers. Satellite is configured with Content Views that will provide a consistent version management of software and repositories used in the solution. Ansible playbooks will then install a Red Hat OpenShift Platform 3.5 cluster consisting of three Master nodes, three Infrastructure Nodes, and three hyper-converged Container-native storage nodes which will also host end-user applications.

2.2. HARDWARE

The focus of this reference architecture is the installation and automation of Red Hat OpenShift Container Platform on HPE ProLiant DL based servers using Red Hat Ansible Tower, Red Hat Satellite, and HPE OneView. The equipment used in this reference architecture was based on the availability of existing lab equipment and may not reflect specific sizing requirements or recommendations for Red Hat OpenShift Container Platform. Refer to the Red Hat OpenShift Container Platform documentation sizing and hardware requirements at *https://access.redhat.com/documentation/en-us/openshift_container_platform/3.5/html/installation_and_configuration/installing-a-cluster#sizing* for guidance on minimum hardware requirements.

The server hardware used in this reference architecture are Hewlett Packard Enterprise (HPE) rack mounted servers, specifically the HPE ProLiant DL360 Gen9 and HPE ProLiant DL380 Gen9 models. The network switches used in this reference architecture are HPE FlexFabric switches, the HPE FF 5930 and HPE FF 5700. The following section provides an overview of the technical specifications for the servers and network equipment used in this reference architecture.

The servers and networking components are shown below in Figure 1. There are two top of rack switches, an HPE FF 5700 and an HPE FF 5930, the HPE FF 5700 is used for management (iLO) and provisioning, and the HPE FF 5930 is used for production.



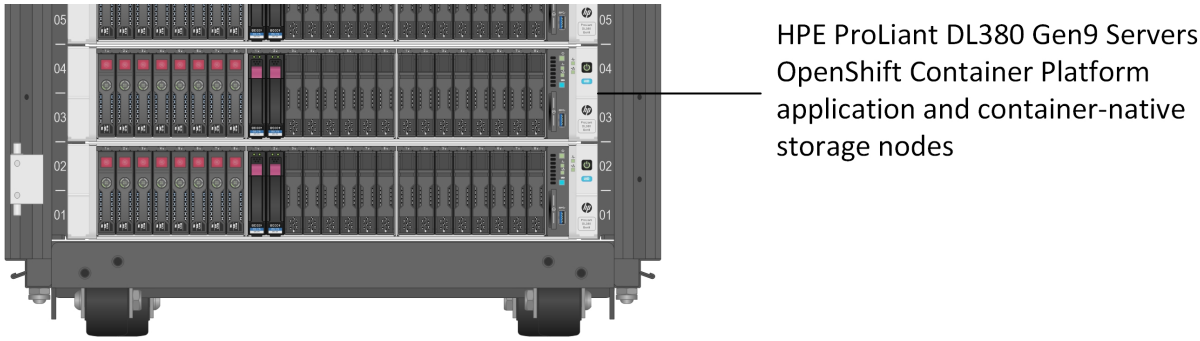


Figure 1: HPE ProLiant DL Rack

2.3. HPE PROLIANT SERVERS

HPE ProLiant DL380 Gen9 Server

The HPE ProLiant DL380 Gen9 is an industry leader in rack mount servers. This reference architecture will leverage the HPE DL380 Gen9 server for the application nodes and will host the Container-native storage.

Specifications:

- Processor family Intel Xeon E5-2600 v3 product family; Intel Xeon E5-2600 v4 product family
- Processor core available 22, 20, 18, 16, 14, 12, 10, 8, 6, or 4
- Maximum memory, 3.0TB; With 128-GB DDR4
- Storage controller (1) Dynamic Smart Array B140i and/or; (1,) Smart Array P440ar; (1) Smart Array P840; Depending on model
- Form factor (fully configured) 2U

Refer to the following link for complete HPE ProLiant DL380 Gen9 server specifications: https://www.hpe.com/us/en/product-catalog/servers/proliant-servers/pip.specifications.hpe-proliant-dl380-gen9-server.7271241.html

Red Hat Container-native storage Nodes HPE ProLiant DL380 Gen9 Configuration

The Red Hat Container-native storage nodes are comprised of three HPE ProLiant DL380 Gen9 servers with the following configuration;

- CPU – 1 x Intel Xeon E5-2643v3 (3.4GHz/6-core/20MB/135W)
- Memory – 64GB Single Rank x4 DDR4-2133
- Storage - 12 HP 1.2TB 6G SAS 10K rpm SFF Drives and 2 HP 400GB 12G SAS Write Intensive SFF Solid State Drives
- Network - HP Ethernet 10Gb 2-port 560FLR-SFP Adapter and ALOM 2 port 10Gb NIC

HPE ProLiant DL360 Gen9 Server

The HPE ProLiant DL360 Gen9 is a 1U rack mount server that will host the Red Hat OpenShift Platform master nodes and the loadbalancer.

Specifications:

- Processor family Intel Xeon E5-2600 v3 product family; Intel Xeon E5-2600 v4 product family

- Processor core available 22, 20, 18, 16, 14, 12, 10, 8, 6, or 4
- Maximum memory 1.5TB
- Storage controller (1) Dynamic Smart Array B140i or; (1) H240ar Host Bus Adapter or; (1) Smart Array P440ar; Depending on model
- Form factor (fully configured) 1U

Refer to the following link for complete HPE ProLiant DL360 Gen9 server specifications: https://www.hpe.com/us/en/product-catalog/servers/proliant-servers/pip.hpe-proliant-dl360-gen9-server.7252836.html

KVM Hypervisor ProLiant DL360 Gen9 Configuration

- Server Platform – HPE ProLiant DL360 Gen9
- CPU – 2 x Intel Xeon E5-2699v3 (2.6GHz/18-core/45MB/145W)
- Memory – 64GB Dual Rank x4 DDR4-2133
- Storage – 2 x HPE 1.2TB 6G SAS 10K rpm SFF (2.5-inch) Drives
- Network - HPE Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter and ALOM 2 port 10Gb NIC

Red Hat OpenShift Container Platform Master Nodes HPE ProLiant DL360 Gen9 Configuration

The Master nodes are deployed on three physical HPE ProLiant DL360 Gen9 servers with the following configuration:

- CPU - 2 x Intel Xeon E5-2699v3 (2.6GHz/18-core/45MB/145W)
- Memory - 64GB Dual Rank x4 DDR4-2133
- Storage - 2 x HPE 1.2TB 6G SAS 10K rpm SFF (2.5-inch) Drives
- Network - HPE Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter and ALOM 2 port 10Gb NIC

HAProxy and Infrastructure Nodes HPE ProLiant DL360 Gen9 Configuration

The load balancer and infrastructure nodes are deployed on three ProLiant DL360 Gen9 servers.

- CPU - 2 x Intel Xeon E5-2690v3 (2.6GHz/12-core/30MB/135W) Processor
- Memory - 256GB Dual Rank x4 DDR4-2133
- Storage - 2 x HPE 1.2TB 6G SAS 10K rpm SFF (2.5-inch) Drives
- Network - HPE Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter and ALOM 2 port 10Gb NIC

2.4. HPE NETWORK

HPE FlexFabric 5930 32QSFP+ Switch

This solution uses the HPE FlexFabric 5930 switch for Red Hat OpenShift Platform production networks. The HPE specification overview for this switch is described below:
"HPE FlexFabric 5930 Series switches are high density, ultra low latency 10 Gigabit Ethernet (GbE) and 40 GbE top of rack (ToR) datacenter switches. This 1U high model has 32x QSFP+ 40GbE ports, 2x power supply slots, and 2x fan tray slots. Power supplies and fan trays must be ordered

separately. The FlexFabric 5930 Switch Series is ideally suited for deployment at the aggregation or server access layer of large enterprise data centers, or at the core layer of medium sized enterprises. These are optimized to meet the increasing requirements for higher performance server connectivity, convergence of Ethernet and storage traffic, the capability to handle virtual environments, and ultra low latency. The FlexFabric product line offers modular core, aggregation, top of rack switching for datacenters."

This description and the full specifications for the HPE FF 5930 can be found at the following link: https://www.hpe.com/us/en/product-catalog/networking/networking-switches/pip.overview.networking-switches.7526493.html

HPE FlexFabric 5700 48G 4XG 2QSFP+ Switch

The iLO management and provisioning networks use the HPE FlexFabric 5700 switch. The HPE specifications overview for the HPE FF 5700 is switch described below:
"HPE FlexFabric 5700 Series switches are cost effective, high density, ultra low latency, top of rack (ToR) data center switches. This model comes with 48x 10/100/1000 ports, 4x fixed 1000 / 10000 SFP+ ports, and 2x QSFP+ for 40 GbE connections. They are ideally suited for deployment at the server access layer of large enterprise data centers. The FlexFabric product line offers modular core, aggregation, top of rack switching for data centers."

This description and the full specifications for the HPE FF 5930 can be found at the following link: https://www.hpe.com/us/en/product-catalog/networking/networking-switches/pip.overview.networking-switches.6638103.html

Physical Network configuration

The network infrastructure is comprised of an HPE FF 5700 and two HPE FF 5930 network switches. The HPE FF 5700 is used for the management (iLO) and provisioning networks. The HPE FF 5930 is used for connectivity between Red Hat OpenShift Container Platform Master, Infrastructure, Application, and Container-native storage nodes.

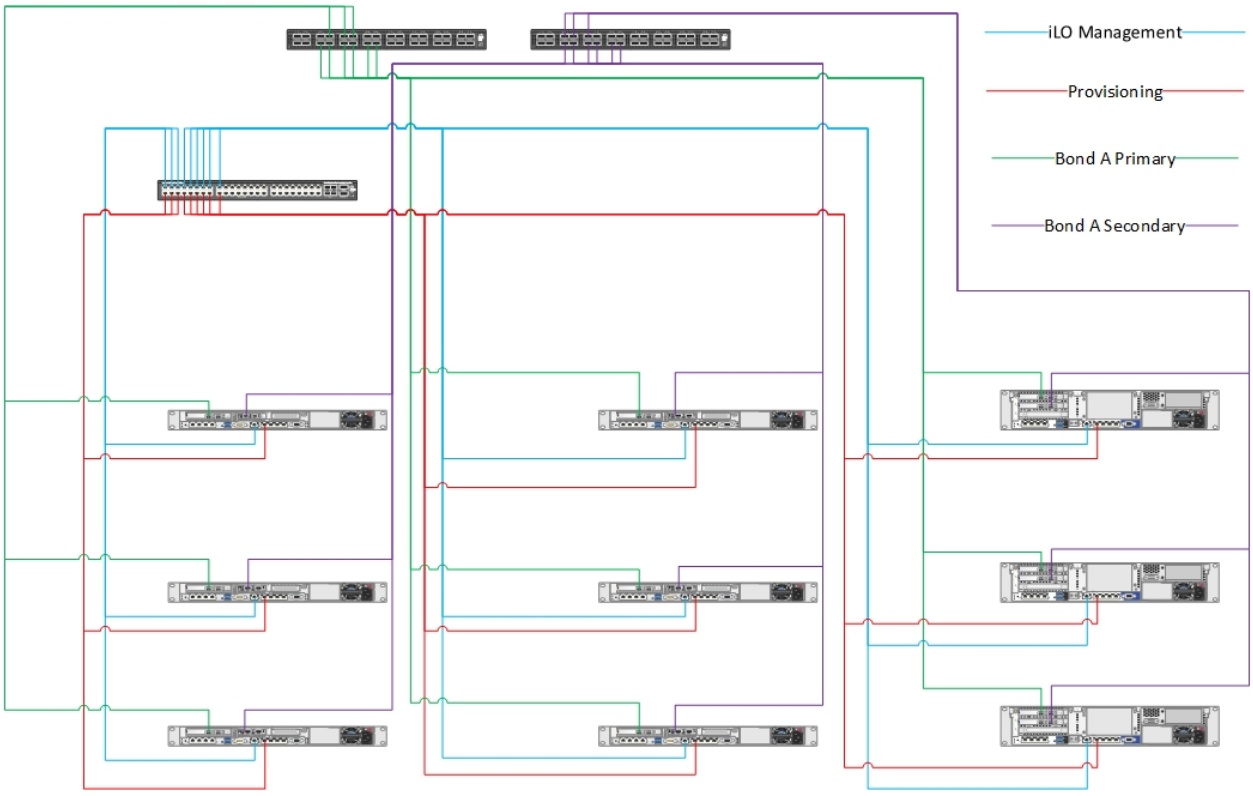


Figure 2: Physical Network

HPE FF 5930 Link Aggregation and Bonding

The HPE FF 5930 is configured with 32 QSFP 40 Gigabit ports. Each port will be split into four 10 Gigabit network ports and connected to 10 Gigabit network interfaces.

As mentioned previously, the HPE FF 5930 used in this reference architecture is configured with 32 Forty Gigabit QSFP ports. Each 40 Gigabit port will be split into four 10 Gigabit ports. The first four 40 Gigabit ports (FortyGigE 1/0/1 - FortyGigE 1/0/4) cannot be split. The first port that can be split is port 5 (FortyGigE 1/0/5). Once the interface has been split, there will be four 10 Gigabit ports labeled:

- XGE1/0/5:1 UP 10G(a) F(a) T 1
- XGE1/0/5:2 UP 10G(a) F(a) T 1
- XGE1/0/5:3 UP 10G(a) F(a) T 1
- XGE1/0/5:4 UP 10G(a) F(a) T 1

The following commands are used to split the FortyGigE interface:

```
[5930-01] interface FortyGigE 1/0/5
[5930-01-FortyGigE1/0/5] using tengige
```

Repeat for interfaces FortyGigE 1/0/6 – FortyGigE 1/0/9, and FortyGigE 2/0/5 - 2/0/9

This will solution will use the Ten Gigabit interfaces shown below in Table 1. Each server is configured with dual port 10 Gigabit Ethernet adapters. The first port in each adapter is connected to the interfaces listed in table 1. An etherent bond is configured between the first port on each 10 Gigabit Ethernet adapter and each port is connected to a different HPE FF 5930 switch.

10Gb Interface 1	10Gb Interface 2
XGE1/0/5:1	XGE2/0/5:1
XGE1/0/5:3	XGE2/0/5:3
XGE1/0/6:1	XGE2/0/6:1
XGE1/0/6:3	XGE2/0/6:3
XGE1/0/7:1	XGE2/0/7:1
XGE1/0/7:3	XGE2/0/7:3

10Gb Interface 1	10Gb Interface 2
XGE1/0/8:1	XGE2/0/8:1
XGE1/0/8:3	XGE2/0/8:3
XGE1/0/9:1	XGE2/0/9:1

Table 1: 10Gb Interfaces

Create the Link Aggregation Groups

Link Aggregation interfaces are configured on the HPE FF 5930 switch.

Link Aggregation Configuration example:

```
interface Bridge-Aggregation11
 port link-type access
 link-aggregation mode dynamic
 lacp edge-port
interface Ten-GigabitEthernet1/0/5:1
 port link-mode bridge
 port link-type access
 port link-aggregation group 11
interface Ten-GigabitEthernet2/0/5:1
 port link-mode bridge
 port link-type access
 port link-aggregation group 11
```

Table 2, shown below, displays the full list of Bridge Aggregations and Interfaces defined on the HPE FF 5930 for the Red Hat OpenShift Container Platform 10 networks.

Network Aggregations	Interface 1	Interface 2
Bridge-Aggregation11	Ten-GigabitEthernet1/0/5:1	Ten-GigabitEthernet2/0/5:1
Bridge-Aggregation21	Ten-GigabitEthernet1/0/5:3	Ten-GigabitEthernet2/0/5:3
Bridge-Aggregation31	Ten-GigabitEthernet1/0/6:1	Ten-GigabitEthernet2/0/6:1
Bridge-Aggregation41	Ten-GigabitEthernet1/0/6:3	Ten-GigabitEthernet2/0/6:3

Network Aggregations	Interface 1	Interface 2
Bridge-Aggregation51	Ten-GigabitEthernet1/0/7:1	Ten-GigabitEthernet2/0/7:1
Bridge-Aggregation61	Ten-GigabitEthernet1/0/7:3	Ten-GigabitEthernet2/0/7:3
Bridge-Aggregation71	Ten-GigabitEthernet1/0/8:1	Ten-GigabitEthernet2/0/8:1
Bridge-Aggregation81	Ten-GigabitEthernet1/0/8:3	Ten-GigabitEthernet2/0/8:3
Bridge-Aggregation91	Ten-GigabitEthernet1/0/9:1	Ten-GigabitEthernet2/0/9:1

Table 2: Bridge Aggregations

2.5. DISK CONFIGURATION

Master and Infrastructure nodes

The Red Hat OpenShift Container Platform Master and Infrastructure nodes are configured with two 1.2 TB SAS Hard Drives. These drives are configured as a RAID1 volume with a single logical drive on the HPE Smart Array controller. The array and logical drive are defined in the HPE OneView server profile applied to these servers. When the server profiles are applied, the array and logical drive is created.

Red Hat Container-native storage Disk Configuration

The Red Hat Container-native storage in this solution is comprised of three HPE ProLiant DL380 Gen9 servers with twelve 1.2 TB Drives and two 400 GB SSD Drives. The twelve 1.2 TB Drives are configured in a RAID6 volume and used for Gluster storage. The two 400 GB Drives are configured in a RAID1 volume with a single logical drive to be used for the operating system. The arrays and logical drives are defined in the HPE OneView server profiles for the Container-native storage servers. They are created when the server profiles are applied to the servers.

2.6. SOFTWARE OVERVIEW

This section provides and overview of the software used in this reference architecture.

Red Hat OpenShift Container Platform 3.5

Red Hat OpenShift Container Platform - based on kubernetes - is used to provide a platform to build and orchestrate the deployment of container-based applications. The platform can run pre-existing container images, or build custom ones directly from source.

Red Hat Satellite

Red Hat Satellite will be used to kickstart the HPE ProLiant DL servers and install Red Hat

Enterprise Linux 7.3 on each of the ProLiant DL servers. Satellite will also provide the software repositories for installing OpenShift Container Platform 3.5 and it's required dependencies. Satellite will be configured to provide DNS and DHCP services for the OpenShift Container Platform environment.

Red Hat Gluster Container-native storage

Red Hat Gluster Container-native storage will provide persistent storage for container based application. Red Hat Gluster Container-native storage will be deployed on the HPE ProLiant DL380 servers. The deployment and configuration will be automated using Ansible playbooks.

Red Hat Ansible Tower

Red Hat Ansible Tower will leverage HPE OneView Ansible playbooks to register the HPE ProLiant DL servers with HPE OneView and apply OneView Server Profiles for the various server roles used in an OpenShift Container Platform deployment; OpenShift Master, OpenShift Application Nodes, and Infrastructure nodes. Ansible Tower will then deploy Red Hat OpenShift Container Platform 3.5 on HPE ProLiant DL servers.

HPE OneView 3.1

HEP OneView 3.1 is used for management and administration of the HPE ProLiant DL servers. OneView is used to manage the ProLiant DL server firmware levels, monitor the health of the servers, and provide remote administration and access to the physical servers. In this reference architecture the OneView 3.1 management appliance is running on a Red Hat 7.3 KVM server. This is the first release of HPE OneView that is supported to run on KVM.

HPE ProLiant Service Pack

HPE ProLiant server firmware and software updates are maintained in the HPE ProLiant Service Pack. The servers were updated with HPE ProLiant Service Pack, 2017.04.0. The ProLiant Service Pack will be maintained as a baseline firmware in HPE OneView and applied to a HPE ProLiant DL server when a server profile is applied.+

Table 3, below shows the firmware versions installed on the ProLiant DL servers used in this reference architecture.

Firmware	Version
iLO	2.50 Sep 23 2016
System ROM	P89 v2.40 (02/17/2017)
Redundant System ROM	P89 v2.30 (09/13/2016)
HP Ethernet 1Gb 4-port 331i Adapter - NIC	17.4.41
HP Ethernet 10Gb 2-port 560FLR-SFP+ Adapter	1.1446.0

Firmware	Version
HPE Ethernet 10Gb 2-port 562SFP+ Adapter	XL710 FW ver
HPE Smart Storage Battery 1 Firmware	1.1
Intelligent Platform Abstraction Data	24.02
Intelligent Provisioning	N/A
Power Management Controller Firmware	1.0.9
Power Management Controller FW Bootloader	1.0
SAS Programmable Logic Device	Version 0x02
Server Platform Services (SPS) Firmware	3.1.3.21.0
Smart Array P440ar Controller	4.52
System Programmable Logic Device	Version 0x34

Table 3: Firmware

ProLiant Service Pack can be found at:
http://h17007.www1.hpe.com/us/en/enterprise/servers/products/service_pack/spp/index.aspx

CHAPTER 3. ANSIBLE

Ansible is a powerful configuration management and orchestration tool, which can be used to automate complicated infrastructure or application deployment tasks.

At it's core, ansible ships with hundreds of modules to manage different host types and software. These hosts can be traditional servers running Red Hat Entperise Linux and other operating systems, as well as an impressive count of network switches and storage arrays. Modules are also included to interact with APIs (application program interface) directly, or wrapped up in a separate module to present a standard set of arguments. For example, the openstack (*os_*) modules are provided to orchestrate an OpenStack cloud.

An ansible *playbook* is built by defining *tasks*. A given *task* will call a *module* with the desired set of arguments. The playbook is then run against an *inventory* which specifies the hosts to run those tasks.

Ansible Tower takes those ansible constructs, and presents them in a clean, modern web GUI. This reference architecture used Ansible Tower by Red Hat version 3.1.1 to orchestrate the provisioning of HPE baremetal ProLiant DL servers, integration with HPE OneView 3.1, installation of Red Hat Enterprise Linux 7.3, and deployment of Red Hat OpenShift Container Platform 3.5 with Container-native storage. This was accomplished using Ansible Tower inventory, templates, roles, tasks,and workflows, to execute the multiple ansible playbooks required for this reference architecture. The installation of Ansible Tower is beyond the scope of this reference architecture. Refer to the Ansible Tower by Red Hat documentation at http://docs.ansible.com/ansible-tower/index.html for complete instructions on the installation and configuration of Ansible Tower by Red Hat.

Dashboard

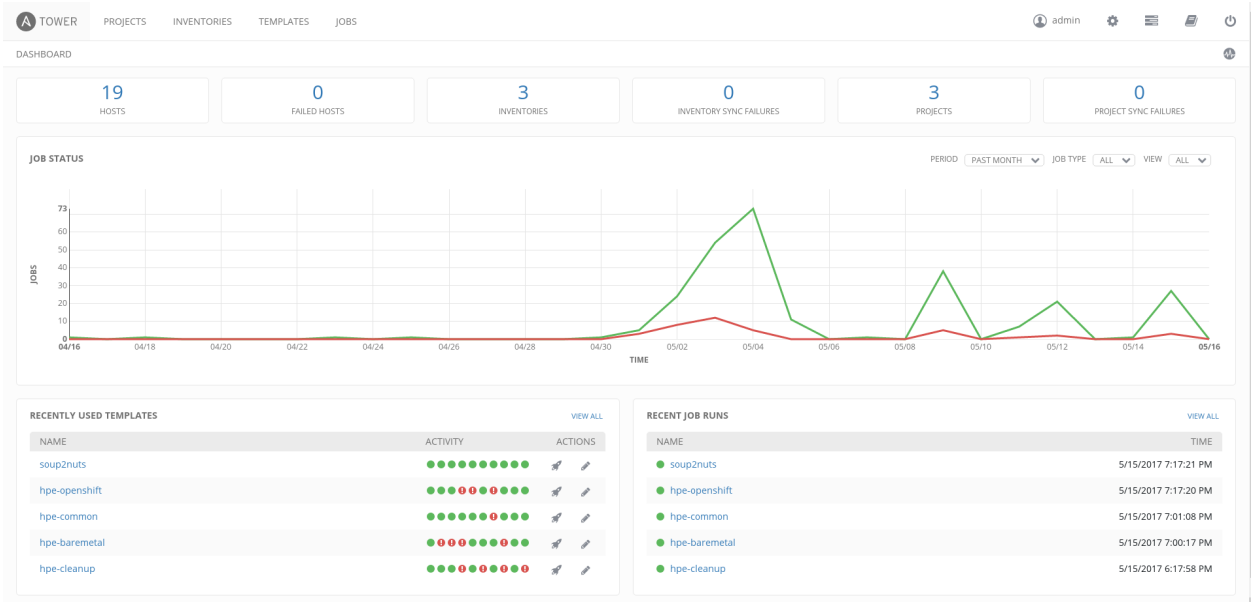


Figure 3: Ansible Tower Dashboard

Inventory

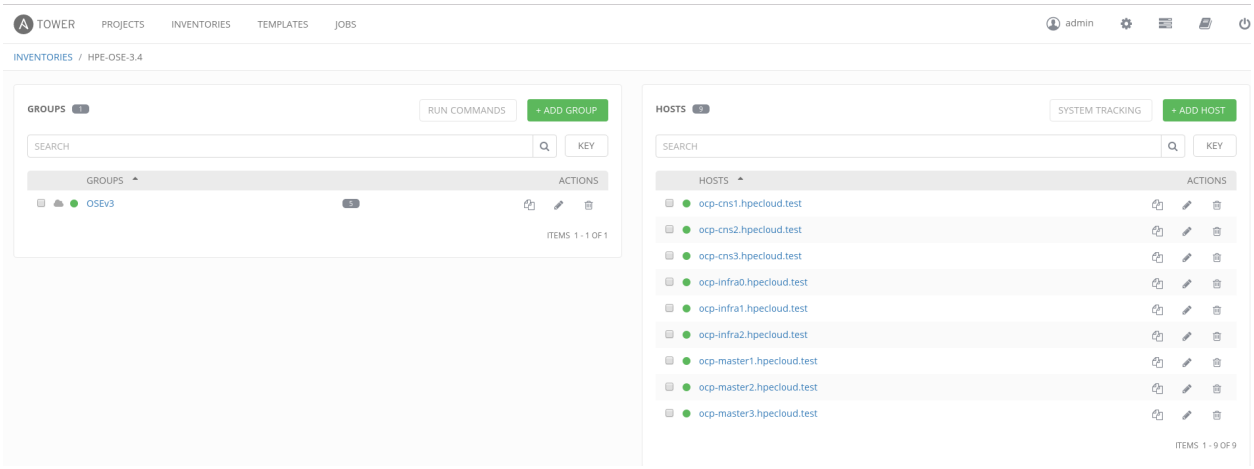


Figure 4: Ansible Tower Inventory

An inventory is a list of targets to run a given playbook against. Inventories can be statically defined, or leverage *dynamic inventories*. A dynamic inventory allows the operator to point Ansible Tower at a source which can change over time, such as a public or private cloud, to generate a list of targets.

Projects

A project refers to a software repository (typically git) where playbooks are stored.

Templates

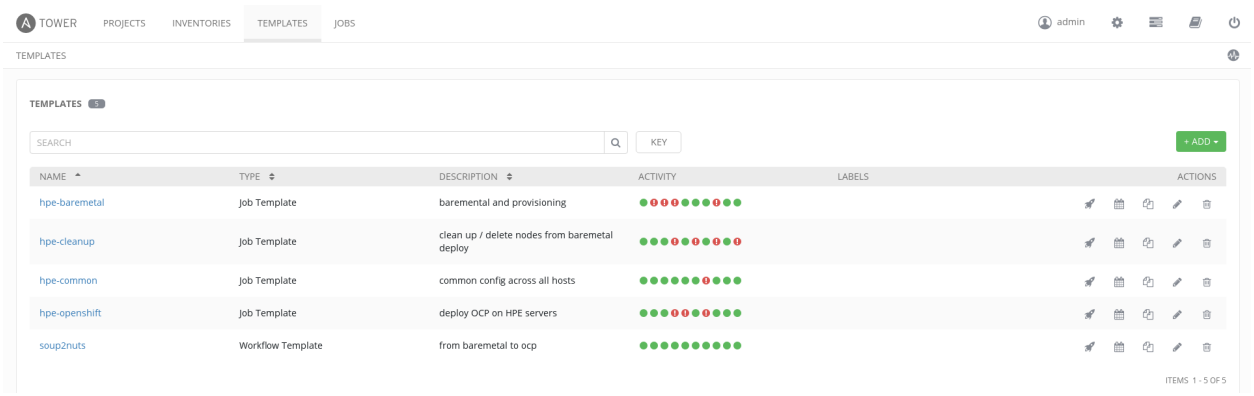


Figure 5: Ansible Tower Templates

Templates tie inventory and projects together. A template will specific which playbook to run from a given project, and execute it against a specific set of hosts from an inventory.

Workflows

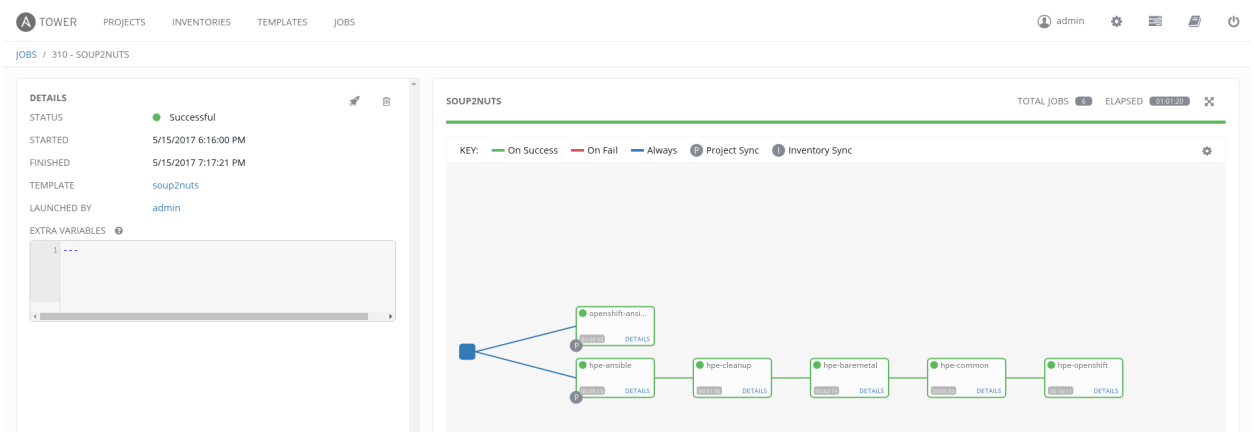


Figure 6: Ansible Tower Workflow

Workflows are a way to chain multiple ansible playbooks together. Different actions can be taken whether a given playbook has succeeded or failed.

A collection of ansible playbooks were created for this reference architecture and synced to Ansible Tower. These playbooks are detailed in a later chapter and available in a github repo.

In addition, the Red Hat OpenShift Container Platform installer is based on Ansible, and an advanced install exposes this to a systems administrator. Inventory and variables can be configured and the playbook run using the standard ansible-playbook command. The use of these playbooks is well documented on the linked web site. This reference architecture only covers integrating and running those playbooks in Tower.

CHAPTER 4. HPE ONEVIEW

OneView is HPE's tool to manage and control many aspects of physical hardware. This includes firmware, BIOS settings, on-board RAID configuration and boot order. OneView supports both blade and rackmount servers. As mentioned previously, this reference architecture is running HPE OneView 3.1 and the OneView management appliance is running on a Red Hat Enterprise Linux 7.3 KVM server. KVM is a newly supported platform for the OneView management appliance with the release of HPE OneView 3.1.

Dashboard

The OneView Dashboard illustrates a highlevel status of all servers, storage, and network equipment registered with OneView. from the Dashboard, an administrator can drill down to find detailed information on alerts.

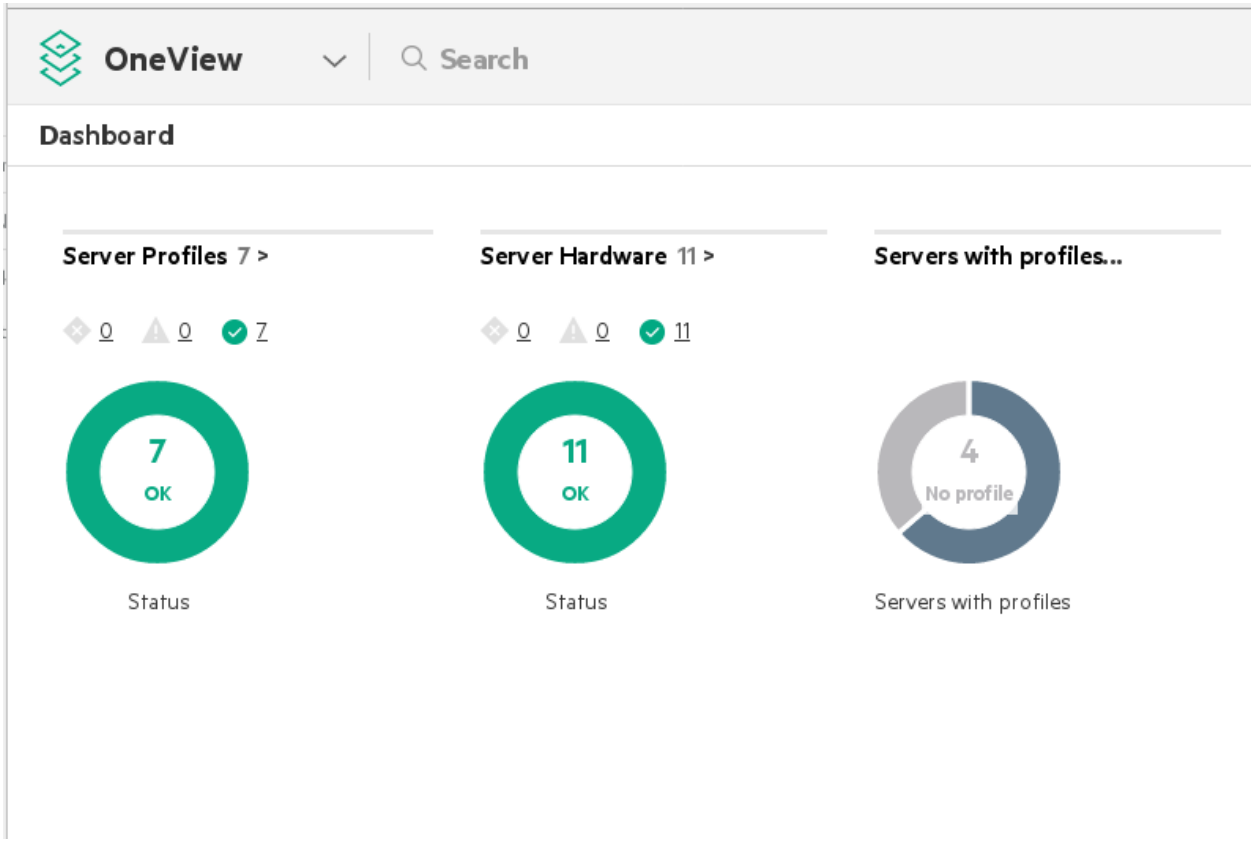


Figure 7: HPE OneView Dashboard

Servers

The servers view shows the servers that have been registered with OneView. Once a server is registered, OneView provides a management interface for the server and reports on the status and health of the server.

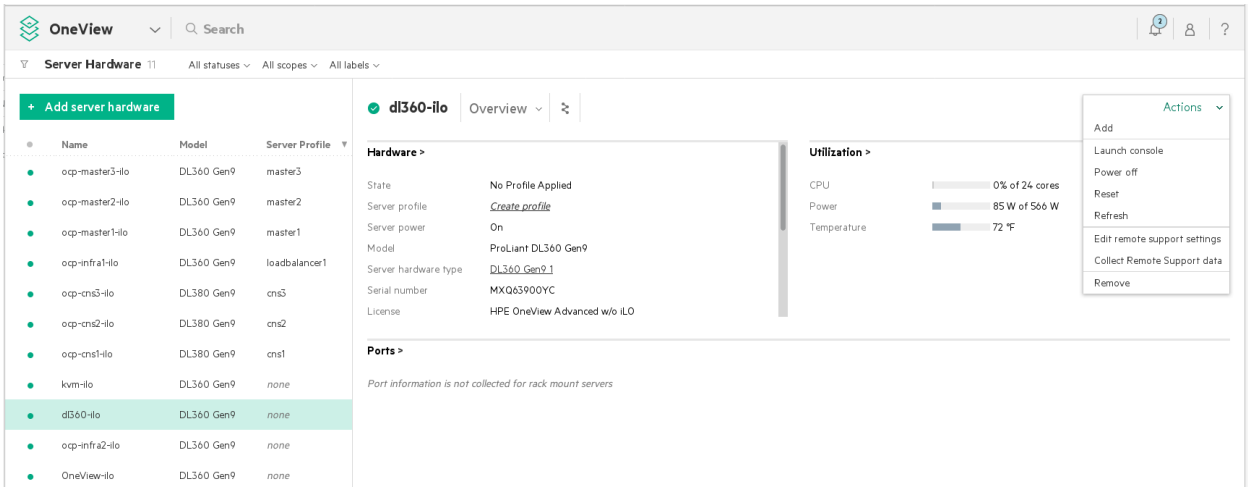


Figure 8: HPE OneView Servers

Server Profiles

Server Profiles are used by OneView to configure a server. When a profile is applied to a server the configuration options specified in the server profile are applied to the server, this includes Firmware levels, BIOS options, boot order, and disk configuration.

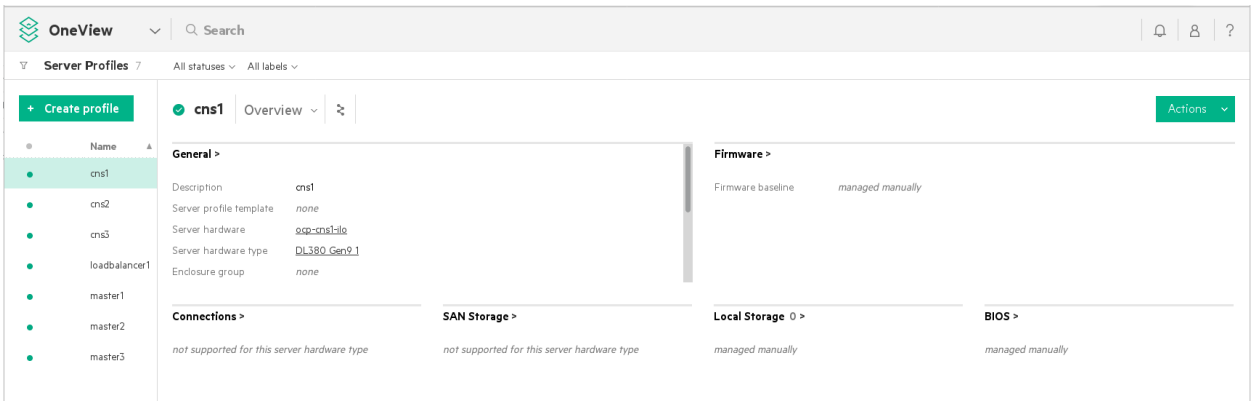


Figure 9: HPE OneView Server Profiles

Firmware Management

OneView provides the ability to store firmware bundles that can then be applied to servers via a server profile. Firmware bundles can be standard bundles that are available through the Service Pack for ProLiant or an administrator can create a custom firmware bundle for specifc configurations. In this reference architecture the firmware bundle used was from the Service Pack for ProLiant version 2017.04.0.

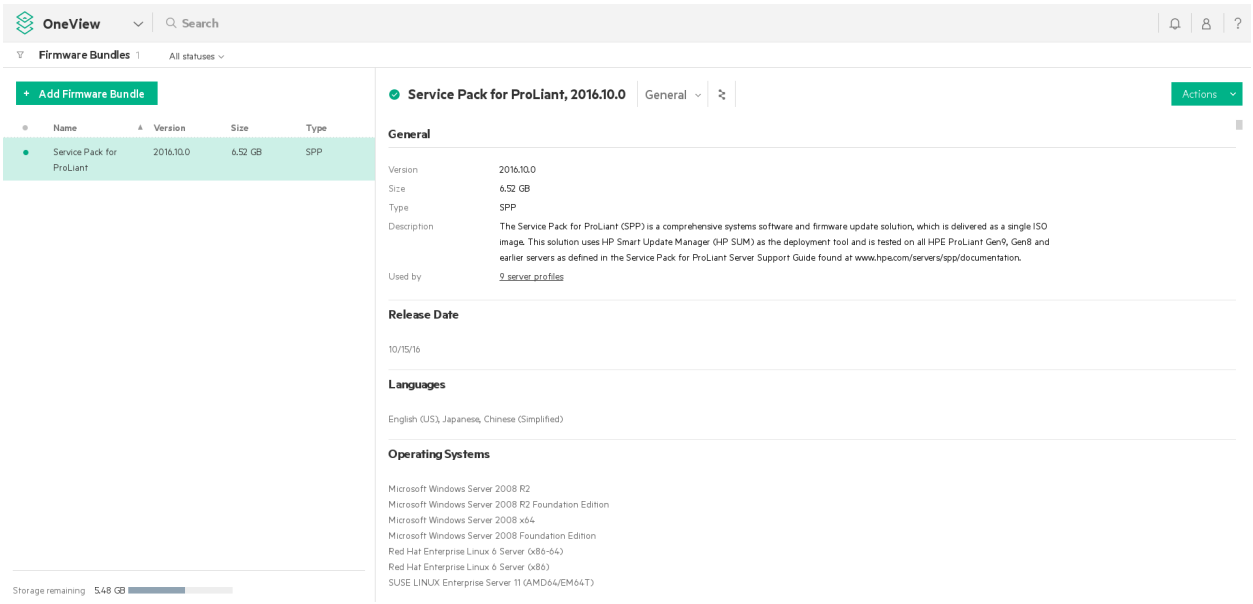


Figure 10: HPE OneView Firmware

OneView Ansible Playbooks

HPE has created a collection of ansible modules to interact with a OneView server. They can be found on github here. There are module for various actions that can be performed via the OneView GUI.

In particular, this reference architecture leverages the modules for the following tasks:

- Import a newly racked server into OneView
- Check for the latest version of available firmware
- Apply a server profile to each node, which:
  - Installs the latest firmware
  - Sets the boot order to use PXE first
  - Configure custom BIOS settings
  - Create one or two RAID volumes, depending on if its a standard or CNS node
- Powers on the server

CHAPTER 5. RED HAT SATELLITE

Red Hat Satellite is the best way to manage your Red Hat infrastructure.

Red Hat Satellite manages the life cycle of an operating system, from initial deployment to ongoing updates. It provides a local mirror of all available Red Hat packages for faster software delivery inside the data center.

A built in DNS server provides automated DNS entry when a host is created, and deletion when the host is decommissioned.

Satellite 6.2 was used in this reference architecture to provision the operating systems, provide access to the required repositories, and to provide DHCP and DNS services. The following sections provide an overview of the Satellite server configuration to support the deployment of Red Hat OpenShift Container Platform 3.5 on baremetal HPE ProLiant DL servers. Installation of the Satellite server is beyond the scope of this document, please refer to the official Satellite document for installation and configuration of Satellite servers.

Configuration to Support OpenShift Container Platform

Repositories

The following repositories are required to be available from the Red Hat Satellite server to deploy OpenShift with Container-native storage:

Name	Repo ID
Red Hat Gluster Storage 3.1 Server (RPMs)	rh-gluster-3-for-rhel-7-server-rpms
Red Hat Enterprise Linux Fast Datapath (RHEL 7 Server) (RPMs)	rhel-7-fast-datapath-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms
Red Hat OpenShift Container Platform 3.5 (RPMs)	rhel-7-server-ose-3.5-rpms
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms
Red Hat Satellite Tools 6.2 (for RHEL 7 Server) (RPMs)	rhel-7-server-satellite-tools-6.2-rpms

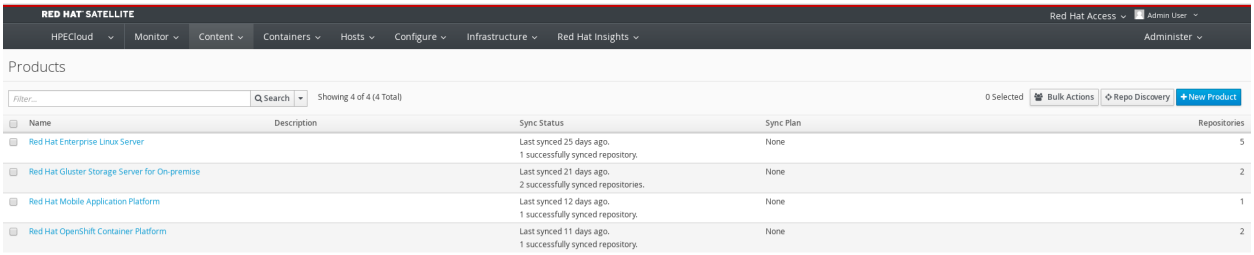


Figure 11: Satellite Products

Content Views

Red Hat Satellite Content views are used to manage the selection of content available to the hosts registered in Satellite. The Content views provide lifecycle management by maintaining repositories with specific software versions available for deployment. In the figure below, titled Satellite Content View, the required yum repositories for OpenShift have been added to the _rhel7-ocp3-5 content view:

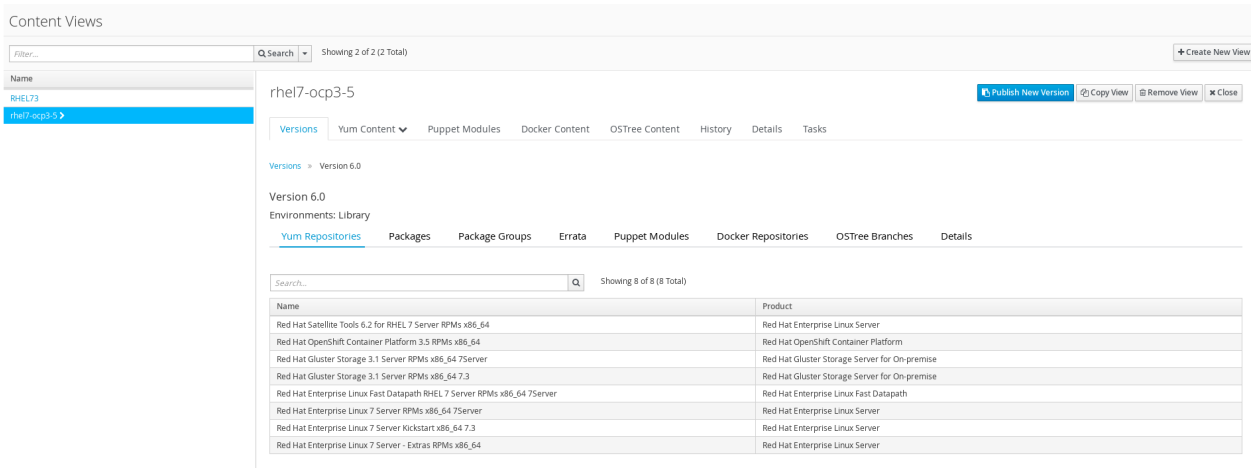


Figure 12: Satellite Content View

Lifecycle Environment

To manage the promotion of content views between development and production, a *lifecycle environment* is used. A content view is published to a specific environment. When that view has been tested and vetted in an environment, it can then be *promoted* to the next level e.g. production.

A lifecycle environment named *ocp-dev* was created and associated with the *rhel7-ocp-3-5* content view.

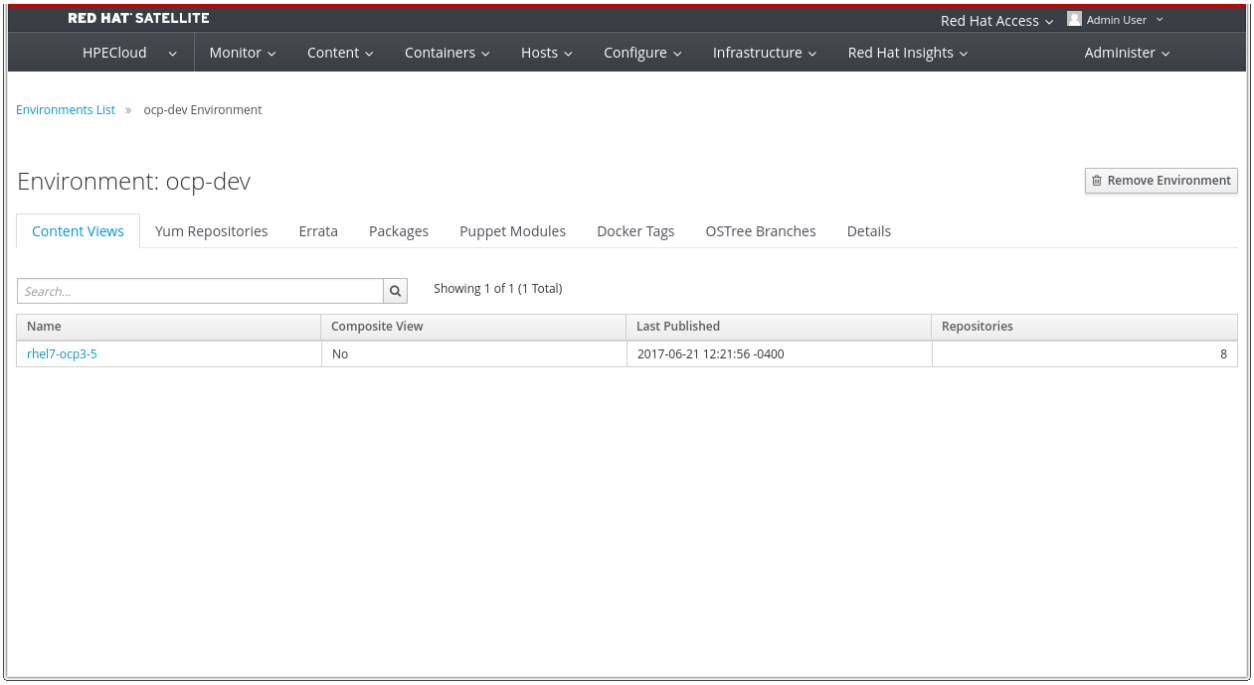


Figure 13: Lifecycle Environment

Subnets

Subnets are defined in Red Hat Satellite so when a host is created, it is allocated an IP from a set range along with the proper netmask and gateway. In this environment, a subnet with the name *hpecloud_ext* was created for the *10.19.20.128/25* network.

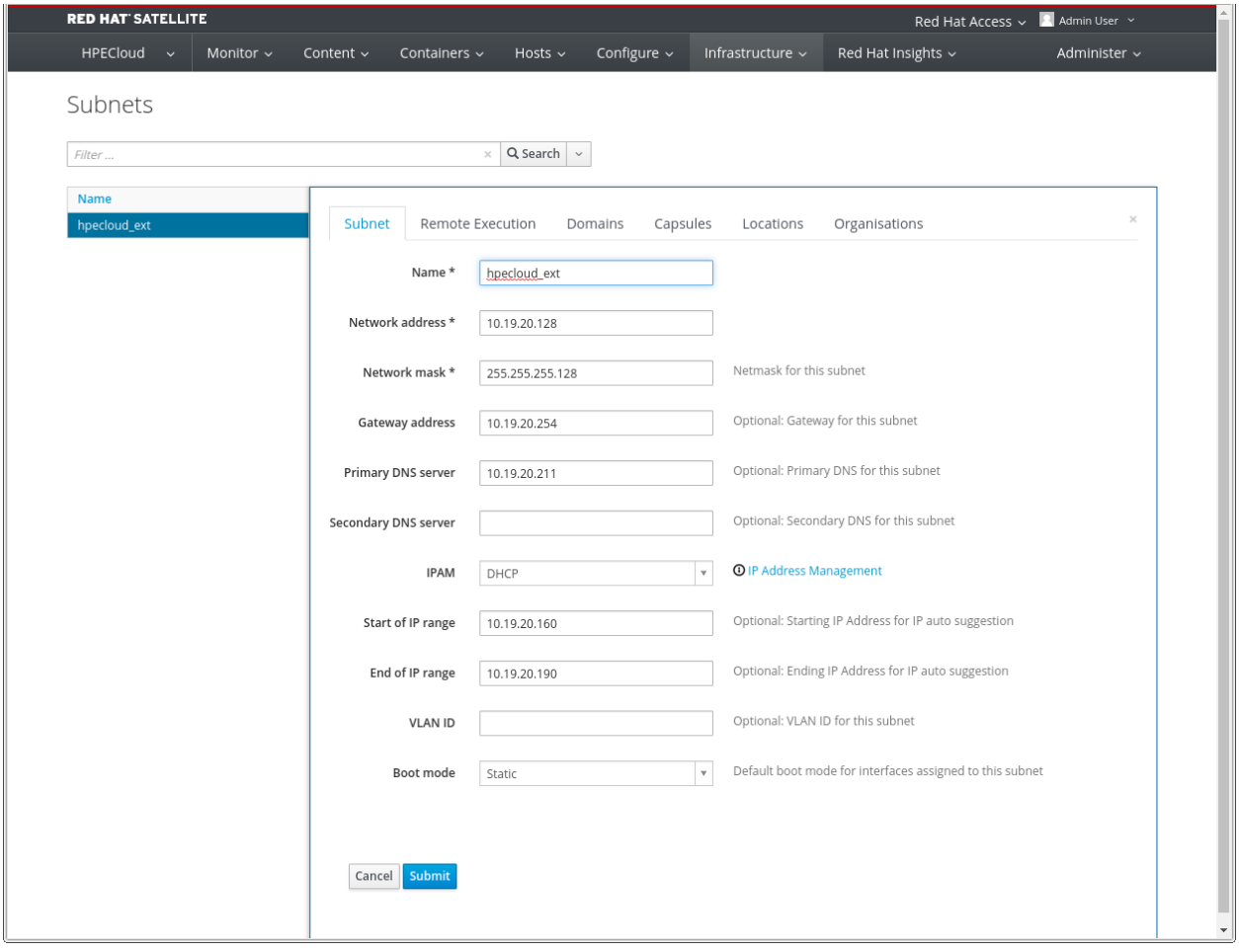


Figure 14: Subnet

Activation Keys

When a new host is deployed for the first time, an activation key is typically supplied. This key is an object in Red Hat Satellite that allows a host to automatically register and attach to any required subscriptions and software channels. An activation called *hpe-ocp* was created so that hosts would have access to the OpenShift Container Platform RPMs.

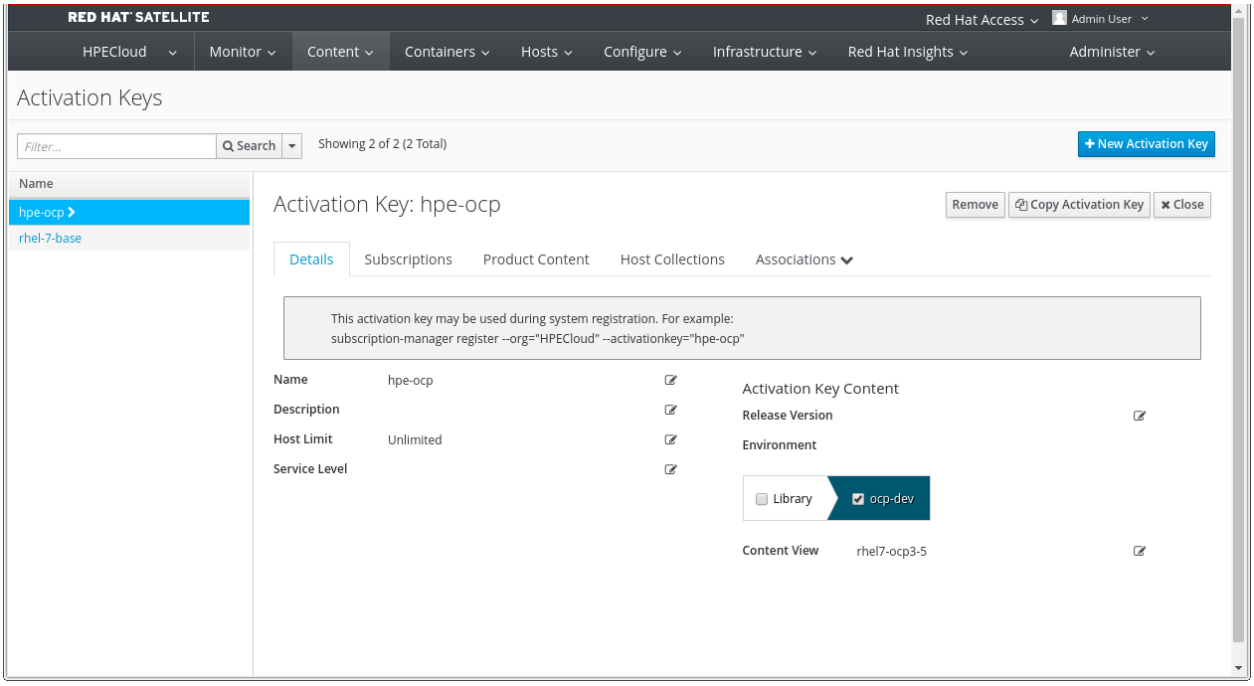


Figure 15: Activation Key Details

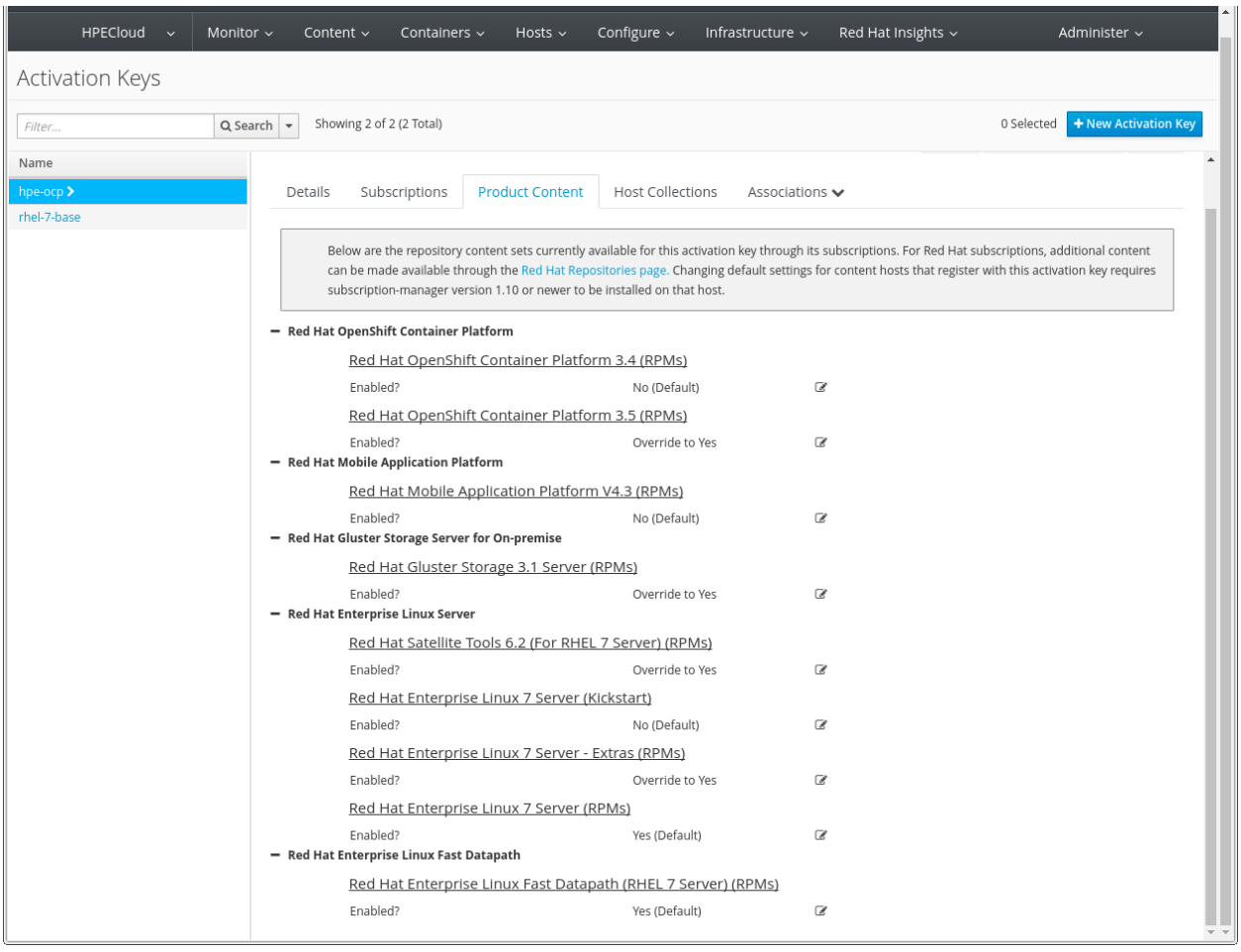


Figure 16: Activation Key Product Content

Kickstart Template

Red Hat Satellite dynamically generates a kickstart configuration for every host that it builds. This allows for operating system installation via PXE boot, using tftp to serve the required kernel and disk image files.

Satellite ships with a number of initial templates. The *Satellite Kickstart Default* template is suitable for most standard installations. To allow Ansible Tower to manage a deployed host, the template must be modified slightly to inject an SSH key.

The *Satellite Kickstart Default* was cloned as *HPE-Kickstart* and modified to install an authorized key for the root user. Below is the relevant portion that was modified:

```
#update local time
echo "updating system time"
/usr/sbin/ntpdate -sub <%= @host.params['ntp-server'] ||
'0.fedora.pool.ntp.org' %>
/usr/sbin/hwclock --systohc

# deploy ansible tower key
mkdir /root/.ssh/
chmod 700 /root/.ssh
echo 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDRcFhowHWK8ZfzTRFZcqs5BbFqryBXWwXKj1HI4dD
ipPTNka0GD6+qeMZiLgNHQ2bn24HXoWSzWRyKSU+cDD5LWpPq9sPRLTO/japC5YQfeOMQbk
SnV4GaglX50oqcI1whSovCXNL0JtxDg8YoWQrhqpM+r3nD+IATOFLeB/kk3Vuc1UHAZvO0W
w9bIw32tK4hOtB2CWsZr3T0xe/k5OZF5v9Y21aiLA//p655N0LrVF08EqOmPQi93EUWTLYv
ZXQyLFuu80PdCIDdhvU1mrQj5iBFDJrQiKSL02zRKr6JDKsvrPyb750R5HsOohEHQlD3KsO
NkJNnzphtVHM1dkf3 dcritch@sputnik.xana.du' >>
/root/.ssh/authorized_keys
echo 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC5/HWM5BBBG+Oj8teXxc0A7bYT7ke6qttnnAm7e90
uaom5tZvjscQHUINo1gvZSv5vMgV4x3Rgm5+/W+3FZEkR9BlymjltWOw5KBu+XvDcJnhUGK
A2gLmaEclQsvB4TVcTv1m3ALa7W0ReqHR64geu638cNLrXiWRXdyttLNq28qQsSIf/m7Pv5
By/jOfmc/xKXbajDuhMa/NsJ1XlHHEljhb2cl/mtkt2TcWht/2nZF1ozAeNDnhHzDLrtmYN
qX0qKyrjF+RHH2t6hBF8iaf/8IxLdgycAxpcY2IPmp2p8RuO4Fs1j4xw/gPwRotYQQ3i0zN
dkdMfr5NTQoBjpRrf root@dc-dev.cloud.lab.eng.bos.redhat.com' >>
/root/.ssh/authorized_keys

<%= snippet "subscription_manager_registration" %>
```

Partition Table

Kickstart templates may include a parition table, but Red Hat Satellite also allows for the creation of custom, dynamic partition tables. For the OpenShift nodes, the OS volume that is configured through HPE OneView is partitioned in Satellite to provide a dedicated docker volume group:

```
<%#
kind: ptable
name: Kickstart default
oses:
- CentOS 5
- CentOS 6
- CentOS 7
- Fedora 16
```

```
- Fedora 17
- Fedora 18
- Fedora 19
- Fedora 20
- RedHat 5
- RedHat 6
- RedHat 7
%>
zerombr
clearpart --all --initlabel
#autopart
#ignoredisk --only-use=sda
part pv.192 --fstype="lvmpv" --ondisk=sda --size=131072
part /boot --fstype="xfs" --ondisk=sda --size=500
volgroup vgos --pesize=4096 pv.192
logvol /  --fstype="xfs" --size=16128 --name=lvroot --vgname=vgos
logvol swap  --fstype="swap" --size=16128 --name=swap --vgname=vgos
logvol /var  --fstype="xfs" --size=65536 --name=lvvar --vgname=vgos
part pv.193 --fstype="lvmpv" --ondisk=sda --size=131072 --grow
volgroup docker-vg --pesize=4096 pv.193
```

The *docker-vg* is later used in the *predeploy* Ansible playbook configure docker storage. More details can be found in the official OpenShift docs.

Host Groups

Common combinations of subnets, operating system and activation keys can be expressed as a *Hostgroup* in Red Hat Satellite. For hosts deployed in this reference architecture, a hostgroup called *hpe-ocp* was created to incorporate the various customizations outlined for host creation.

CHAPTER 6. WORKFLOW AND PLAYBOOK DEEP DIVE

This section provides a detailed walk-through of the Ansible playbooks created for this reference architecture and how the Ansible Tower Workflow ties it all together.

6.1. OVERVIEW

The set of playbooks were created with modularity in mind. Rather than create one monolithic playbook to interact with the various services, a role was created for each major step. This makes debugging sections easier, as well as allowing for re-use of certain roles in other infrastructures. For instance, the provisioning playbook could be used in another Ansible Tower workflow, with the OpenShift Container deployment swapped out for another infrastructure stack.

The Red Hat OpenShift Container Platform installer is based on Ansible, and an advanced install exposes this to a systems administrator. Inventory and variables can be configured and the playbook can be run using the standard ansible-playbook command. The use of these playbooks is well documented on the OpenShift site. This reference architecture only covers integrating and running those playbooks in Tower.

Playbooks to handle all of the provisioning steps leading up to the OpenShift installation are available in a github repo. Some modifications will be required to work in different environments, outlined below.

6.2. INVENTORY AND VARIABLES

The reference architecture playbooks require an inventory file and environment specific variables to be set. This will need to be customized for every unique deployment. Since the openshift-ansible playbooks also require an inventory, that inventory is extended to provide the required information for the provisioning playbooks.

ansible-hosts

```
## required for provisoning
ocp-master1.hpecloud.test ansible_host=192.168.1.173
ilo_ip=192.168.1.136
ocp-master2.hpecloud.test ansible_host=192.168.1.174
ilo_ip=192.168.1.137
ocp-master3.hpecloud.test ansible_host=192.168.1.175
ilo_ip=192.168.1.138
ocp-infra0.hpecloud.test ansible_host=192.168.1.172
ilo_ip=192.168.1.135
ocp-infra1.hpecloud.test ansible_host=192.168.1.170
ilo_ip=192.168.1.133
ocp-infra2.hpecloud.test ansible_host=192.168.1.171
ilo_ip=192.168.1.134
ocp-cns1.hpecloud.test ansible_host=192.168.1.176 ilo_ip=192.168.1.140
ocp-cns2.hpecloud.test  ansible_host=192.168.1.177 ilo_ip=192.168.1.141
ocp-cns3.hpecloud.test  ansible_host=192.168.1.178 ilo_ip=192.168.1.142
[cns]
ocp-cns1.hpecloud.test
ocp-cns2.hpecloud.test
ocp-cns3.hpecloud.test
```

```
## standard openshift-ansible inventory
[OSEv3:children]
masters
nodes
etcd
lb
nfs

[masters]
ocp-master1.hpecloud.test
ocp-master2.hpecloud.test
ocp-master3.hpecloud.test

[etcd]
ocp-master1.hpecloud.test
ocp-master2.hpecloud.test
ocp-master3.hpecloud.test


[lb]
ocp-infra0.hpecloud.test
[nfs]
ocp-infra0.hpecloud.test

[nodes]
ocp-master1.hpecloud.test
ocp-master2.hpecloud.test
ocp-master3.hpecloud.test
ocp-infra1.hpecloud.test openshift_node_labels="{'region': 'infra',
'zone': 'west'}"
ocp-infra2.hpecloud.test openshift_node_labels="{'region': 'infra',
'zone': 'east'}"
ocp-cns1.hpecloud.test  openshift_node_labels="{'region': 'primary',
'zone': 'east'}"
ocp-cns2.hpecloud.test  openshift_node_labels="{'region': 'primary',
'zone': 'west'}"
ocp-cns3.hpecloud.test  openshift_node_labels="{'region': 'primary',
'zone': 'west'}"
```

The standard openshift-ansible inventory is extended by adding an entry for each host, with two variables that must be set:

- **ansible_host** the desired static IP of the deployed servers.
- **ilo_ip** the IP for the node's HPE Integrated Lights Out Interface

In addition, the nodes that are to be used for Red Hat Container-native storage are declared in a *[cns]* group.

There are also three files for variables:

- **group_vars/all** common variables for all nodes
- **group_vars/cns** specific disk configuration for storage nodes
- **groups_vars/OSEv3** OpenShift variables

group_vars/all contains variables common across all nodes:

group_vars/all

```
---
## satellite vars (change to match target environment)
satellite_user: "admin"
satellite_url: "https://192.168.1.211"
location_id: 2
organization_id: 1
environment_id: 1
hostgroup_id: 1
ansible_domain: hpecloud.test
ntp_server: "clock.corp.redhat.com"
rndc_key: "r/24yYpTOcnIxvXul+xz3Q=="
ilo_username: "Administrator"
bios_settings:
- id: "CustomPostMessage"
  value: "!! automate the planet !!"
- id: "MinProcIdlePkgState"
  value: "NoState"
- id: "EnergyPerfBias"
  value: "MaxPerf"
- id: "PowerProfile"
  value: "MaxPerf"
- id: "IntelQpiPowerManagement"
  value: "Disabled"
- id: "PowerRegulator"
  value: "StaticHighPerf"
- id: "MinProcIdlePower"
  value: "NoCStates"
storage_cfg:
  controllers:
    - deviceSlot: "Embedded"
      importConfiguration: false
      initialize: true
      mode: "RAID"
      logicalDrives:
      - bootable: true
        driveNumber: 1
        driveTechnology: "SasHdd"
        name: "os"
        numPhysicalDrives: 2
        raidLevel: "RAID1"
## oneview vars
oneview_auth:
  ip: "192.168.1.233"
  username: "Administrator"
  api_version: 300
```

Notice that a *storage_cfg* is defined which specifies how the disks should be configured. The *[cns]* nodes have additional disks for gluster, so they have a different *storage_cfg* variable set to create the additional RAID volume. This is configured in the *cns* group variables file:

group_vars/cns

```
---
```

```
storage_cfg:
  controllers:
    - deviceSlot: "Embedded"
      importConfiguration: false
      initialize: true
      mode: "RAID"
      logicalDrives:
      - bootable: true
        driveNumber: 1
        driveTechnology: "SasSsd"
        name: "os"
        numPhysicalDrives: 2
        raidLevel: "RAID1"
      - bootable: false
        driveNumber: 2
        driveTechnology: "SasHdd"
        name: "cns"
        numPhysicalDrives: 12
        raidLevel: "RAID6"
```

A number of variables need to be configured for the Red Hat OpenShift Container Platform deployment. These can be customized for an system administrator's particular needs. The following are the settings used in this reference architecture:

group_vars/OSEv3

```
ansible_ssh_user: root
deployment_type: openshift-enterprise
openshift_master_identity_providers: [{'name': 'htpasswd_auth',
'login': 'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
openshift_master_cluster_method: native
openshift_master_cluster_hostname: openshift-master.hpecloud.test
openshift_master_cluster_public_hostname: openshift-
master.hpecloud.test
openshift_use_openshift_sdn: true
openshift_hosted_registry_storage_kind: nfs
openshift_hosted_registry_storage_access_modes: ['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory: /exports
openshift_hosted_registry_storage_nfs_options: '*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name: registry
openshift_hosted_registry_storage_volume_size: 10Gi
openshift_hosted_metrics_deploy: false
openshift_hosted_logging_deploy: false
openshift_master_default_subdomain: paas.hpecloud.test
openshift_master_htpasswd_users: {'davidc':
'$apr1$znPqASZl$WjOV1pFe4diJJPhjZgW2q1', 'kbell':
'$apr1$.WBeJqbh$aic2L/5dxbnkdoEC0UWiT.', 'gluster':
'$apr1$OMdwuyb6$bF2f3hSfwsE9XOyCaFEOP.'}
```

The *openshift_master_htpasswd_users* parameter determins a set of cluster users to create during the installation. It is a dict of username: password pairs. The passwords can be generated with htpasswd:

```
$ htpasswd -n gluster
New password:
Re-type new password:
gluster:$apr1$OMdwuyb6$bF2f3hSfwsE9XOyCaFEOP.
```

Finally, the provisioning playbooks rely on a number passwords and keys:

- ILO Administrator on target nodes
- Satellite Administrator
- Root password on target nodes
- Gluster admin password created above
- RNDC key for the DNS server

For security reasons, these passwords are encrypted using Ansible Vault. To create a vaulted password file:

- modify passwords.yaml to match the target environment:

```
---
- name: set ilo password
  set_fact:
    ilo_pw: "P@ssw0rd"
- name: set oneview password
  set_fact:
    oneview_pw: "P@ssw0rd"
- name: set satellite password
  set_fact:
    satellite_pw: "P@ssw0rd"
- name: set root password
  set_fact:
    root_pw: "P@ssw0rd"
- name: set gluster admin password
  set_fact:
    gluster_pw: "n0tP@ssw0rd"
- name: set rndc_key
  set_fact:
    rndc_key: "r/42yYsCOcnIxvXul+xz3Q=="
```

- encrypt the file with *ansible-vault* and store it under *roles/passwords/tasks/main.yaml*:

```
$ ansible-vault encrypt passwords.yaml --
output=roles/passwords/tasks/main.yaml
```

This role is then called by any other roles that require these credentials.

6.3. WORKFLOW

Ansible Tower supports the creation of *workflows*, allowing multiple job templates to be chained together. For this reference architecture, a workflow entitled *hpe-end2end* was created. This workflow calls the following job templates:

- hpe-cleanup

- hpe-provisioning
- hpe-common
- hpe-predeploy
- hpe-openshift
- hpe-cns

Each playbook is annotated below.

6.4. PLAYBOOKS

hpe-provisioning

Although this is not the first playbook in the workflow, the cleanup playbook will make more sense after working through this one.

The *provisioning* playbook includes the *provisioning* role:

playbooks/provisioning.yaml

```
---
## bare metal provisioning

- name: get passwords
  hosts: all
  remote_user: root
  gather_facts: false
  roles:
    - ../roles/passwords

- name: baremetal provisioning
  hosts: all
  remote_user: root
  gather_facts: false
  roles:
    - ../roles/provisioning
  environment:
    PYTHONPATH: "$PYTHONPATH:/usr/share/ansible"
    ONEVIEWSDK_IP: "{{ oneview_auth.ip }}"
    ONEVIEWSDK_USERNAME: "{{ oneview_auth.username }}"
    ONEVIEWSDK_PASSWORD: "{{ oneview_pw }}"
    ONEVIEWSDK_API_VERSION: "{{ oneview_auth.api_version }}"
```

The following is the directory structure of that role:

hpe-provisioning

```
├── roles
│   └── provisioning
│       └── tasks
│           ├── addserver.yaml
│           ├── facts.yaml
```

```
│           ├── firmware.yaml
│           ├── main.yaml
│           ├── poweron.yaml
│           ├── profiles.yaml
│           └── satellite.yaml
```

main.yaml is the entry point for the role, which includes a set of playbooks to execute the required tasks:

roles/provisioning/tasks/main.yaml

```
---
# set some vars
- include: ./facts.yaml
# add the server to oneview
- include: ./addserver.yaml
# get firmware version
- include: ./firmware.yaml
# create profile templates -> profiles
- include: ./profiles.yaml
# create the host in satellite
- include: ./satellite.yaml
# power on server to kickstart over pxe
- include: ./poweron.yaml
```

The first included set of tasks is *facts.yaml* which sets a fact for the ILO hostname based on the node's hostname, and uses the *hpilo_facts* module to generate the MAC addresses of the provisioning and bonded interfaces. In this case, *eth0* is the interface used for provisioning while *eth4* and *eth6* are 10 gig interfaces bonded together for all other traffic. Note that the task assumes the ILO hostname is node's hostname with *-ilo* appended to the end. For example, **ocp-infra1**'s ILO hostname would be **ocp-infra1-ilo**.

roles/provisioning/tasks/facts.yaml

```
---
- set_fact:
    ilo_name: "{{ inventory_hostname.split('.').0 }}-ilo"
  tags:
  - ilo

- name: get facts from ilo
  hpilo_facts:
    host: "{{ ilo_ip }}"
    login: "{{ ilo_username }}"
    password: "{{ ilo_pw }}"
  register: ilo_facts
  delegate_to: localhost
  tags:
  - ilo

- name: set PXE MAC address
  set_fact:
    pxe_mac: "{{ ilo_facts.ansible_facts.hw_eth0.macaddress }}"
  tags:
```

```
  - ilo

## IP + 20, disabled after kickstart
- name: set PXE IP of new host
  set_fact:
    pxe_ip: "{{ ansible_host.split('.').0 }}.{{
ansible_host.split('.').1 }}.{{ ansible_host.split('.').2 }}.{{
ansible_host.split('.').3|int + 20 }}"

- name: set slave0 MAC address
  set_fact:
    slave0_mac: "{{ ilo_facts.ansible_facts.hw_eth4.macaddress }}"
  tags:
  - ilo

- name: set slave1 MAC address
  set_fact:
    slave1_mac: "{{ ilo_facts.ansible_facts.hw_eth6.macaddress }}"
  tags:
  - ilo
```

addserver.yaml first checks to see if the host is already in HPE OneView or not via the *oneview_server_hardware_facts* module. It adds the server with *oneview_server_hardware* if it does not exist. Later tasks rely on some facts provided by the OneView modules. If the host exists, those facts are looked up. Otherwise, they are captured with the *register* argument when added with the *oneview_server_hardware* module.

roles/provisioning/tasks/addserver.yaml

```
---
- name: check if server is in oneview
  oneview_server_hardware_facts:
    name: "{{ ilo_name }}"
  register: server_facts_exists
  delegate_to: localhost
  tags:
  - mkserver

- set_fact:
    server_exists: false
  when:
  - server_facts_exists.ansible_facts.server_hardwares|length == 0
  tags:
  - mkserver
- set_fact:
    server_exists: true
  when:
  - server_facts_exists.ansible_facts.server_hardwares|length > 0
  tags:
  - mkserver

- name: add a server to oneview
  oneview_server_hardware:
    state: present
    data:
```

```
      hostname: "{{ ilo_ip }}"
      username: "{{ ilo_username }}"
      password: "{{ ilo_pw }}"
      force: false
      licensingIntent: "OneViewNoiLO"
      configurationState: "Managed"
  register: server_facts_new
  delegate_to: localhost
  tags:
  - mkserver
  when:
  - server_exists == false
- set_fact:
    server_facts: "{{ server_facts_new.ansible_facts.server_hardware
}}"
  tags:
  - mkserver
  when:
  - server_exists == false
- set_fact:
    server_facts: "{{
server_facts_exists.ansible_facts.server_hardwares.0 }}"
  tags:
  - mkserver
  when:
  - server_exists == true
```

The next playbook, *firmware.yaml* simply queries HPE OneView with the *oneview_firmware_driver_facts* module to determine the latest available firmware. The URI for the firmware is saved so it can be used when applying HPE OneView server profiles.

roles/provisioning/tasks/firmware.yaml

```
---
- name: get fw driver info
  oneview_firmware_driver_facts:
    params:
      sort: 'releaseDate:descending'
      start: 0
      count: 1
  register: fw_info
  delegate_to: localhost
  tags:
  - firmware
- name: set fw version
  set_fact:
    fw_baseline_uri: "{{ fw_info['ansible_facts']
['firmware_drivers'].0['uri'] }}"
  tags:
  - firmware
```

profiles.yaml takes all the facts generated along the way, in addition to some of the pre-defined variables, to apply a profile in HPE OneView. When the profile is applied, HPE OneView will boot the server to apply any BIOS and storage settings as well as check the firmware level and update as required. This is one of the longer running plays, taking approximately 20 minutes to complete.

roles/provisioning/tasks/profiles.yaml

```
---
- set_fact:
      model: "{{ server_facts.model }}"
      short_model: "{{ server_facts.shortModel }}"
      dl_model: "{{ server_facts.shortModel.split(' ').0 }}"
  tags:
  - model

- name: create a service profile and assign it to a node
  oneview_server_profile:
    state: present
    data:
      name: "{{ inventory_hostname.split('.').0 }}"
      server_hardware: "{{ ilo_name }}"
      description: "OpenShift Nodes - {{ short_model }}"
      serverHardwareTypeName: "{{ short_model }} 1"
      boot:
        manageBoot: true
        order: ["PXE", "CD", "USB", "HardDisk"]
      bootMode:
        manageMode: true
        mode: "BIOS"
        pxeBootPolicy: null
      bios:
        manageBios: true
        overriddenSettings: "{{ bios_settings }}"
      firmware:
        firmwareBaselineUri: "{{ fw_baseline_uri }}"
        #firmwareInstallType: "FirmwareOnly"
        firmwareInstallType: "FirmwareOnlyOfflineMode"
        forceInstallFirmware: false
        manageFirmware: true
      localStorage: "{{ storage_cfg }}"
  register: output
  delegate_to: localhost
  tags:
  - templates
```

Once the server is properly managed by HPE OneView, it is deployed against the Red Hat Satellite server. The *uri* module is used to interact with Red Hat Satellite's API. Satellite is first queried to check if the host exists already, and adds it if necessary. The *pxe_mac* fact generated earlier along with the *ansible_host* IP defined in the inventory configures the NIC in Satellite and creates a static DNS entry.

The URI call will create the host with one provisioning interface, and a bonded pair of 10gig NICs. If this configuration differs from the target environment, then *interfaces_attributes* below should be updated to match.

roles/provisioning/tasks/satellite.yaml

```
- name: "Get Host ID from Satellite"
  uri:
    url: "{{ satellite_url }}/api/hosts/?search=name={{
```

```
inventory_hostname }}"
    user: "{{ satellite_user }}"
    password: "{{ satellite_pw }}"
    headers:
      Content-Type: "application/json"
      Accept: "application/json"
    force_basic_auth: yes
    validate_certs: False
    body_format: json
    return_content: yes
    status_code: 200
  ignore_errors: false
  delegate_to: localhost
  register: check_host_response
  tags:
  - satellite
- debug:
    var: check_host_response
    verbosity: 2

- set_fact:
    host_in_satellite: true
  when:
  - check_host_response.json.subtotal == 1
  tags:
  - satellite
- set_fact:
    host_in_satellite: false
  when:
  - check_host_response.json.subtotal == 0
  tags:
  - satellite

- name: "Add Host to Satellite"
  uri:
    url: "{{ satellite_url }}/api/hosts/"
    method: POST
    user: "{{ satellite_user }}"
    password: "{{ satellite_pw }}"
    headers:
      Content-Type: "application/json"
      Accept: "application/json"
    force_basic_auth: yes
    validate_certs: False
    return_content: yes
    body_format: json
    body:
      host:
        name: "{{ inventory_hostname }}"
        location_id: "{{ location_id }}"
        organization_id: "{{ organization_id }}"
        environment_id: "{{ environment_id }}"
        hostgroup_id: "{{ hostgroup_id }}"
        build: true
        enabled: true
        managed: true
```

```
        root_pass: "{{ root_pw }}"
        overwrite: true
        interfaces_attributes:
          -
            mac: "{{ pxe_mac }}"
            primary: false
            type: "interface"
            identifier: "pxe0"
            provision: true
            ip: "{{ pxe_ip }}"
            managed: true
            subnet_id: 1
            domain_id: 1
          -
            mac: "{{ slave0_mac }}"
            primary: true
            type: "bond"
            ip: "{{ ansible_host }}"
            mode: "active-backup"
            identifier: "bond0"
            attached_devices: "slave0,slave1"
          -
            mac: "{{ slave0_mac }}"
            primary: false
            type: "interface"
            identifier: "slave0"
          -
            mac: "{{ slave1_mac }}"
            primary: false
            type: "interface"
            identifier: "slave1"
    status_code: 201
  ignore_errors: false
  delegate_to: localhost
  register: add_host_response
  when:
  - host_in_satellite == false
  tags:
  - satellite
```

Finally, the server can be powered on. Since PXE was configured as the first boot option for each server in HPE OneView, it will begin the kickstart process when first powered up. A wait is declared so the playbook will not exit until the server is online and responding to SSH, e.g. fully provisioned. Putting in the check that the server is up helps ensure the subsequent playbooks are executed when the host is installed and available:

roles/provisioning/tasks/poweron.yaml

```
---
- name: power on server
  oneview_server_hardware:
    state: power_state_set
    data:
      name: "{{ ilo_name }}"
      powerStateData:
```

```
        powerState: "On"
        powerControl: "MomentaryPress"
  register: output
  delegate_to: localhost
  tags:
  - poweron
- name: wait for server to become available
  local_action: wait_for port=22 host='{{ ansible_host }}' delay=30
timeout=3600
  tags:
  - poweron
```

hpe-common

hpe-common contains tasks that are common to any environment. It includes the type of things one would bake in to their 'golden image.' Examples would be the creation of users, or installation of packages.

The *common* playbook calls the *common* role:

playbooks/common.yaml

```
---
## common tasks for all servers

- name: common
  hosts: all
  remote_user: root
  roles:
    - ../roles/common
```

The role is laid out as follows:

```
├── roles
│   ├── common
│   │   ├── handlers
│   │   │   └── main.yaml
│   │   ├── tasks
│   │   │   ├── main.yaml
│   │   │   ├── motd.yaml
│   │   │   ├── ntp.yaml
│   │   │   └── packages.yaml
│   │   └── templates
│   │       ├── chrony.j2
│   │       └── motd.j2
```

And the role is executed in the desired order:

roles/common/tasks/main.yaml

```
---
- include: packages.yaml
  tags:
  - packages
- include: ntp.yaml
```

```
  tags:
  - ntp
- include: motd.yaml
  tags:
  - motd
```

packages.yaml installs some base packages:

roles/common/tasks/packages.yaml

```
---
- name: install common packages
  yum:
    name: '{{ item }}'
    state: latest
  tags:
    - rhn
  with_items:
  - screen
  - tmux
  - nfs-utils
  - sg3_utils
  - policycoreutils-python
  - '@network-file-system-client'
```

The NTP service is configured:

roles/common/tasks/ntp.yaml

```
---
- name: install chrony
  yum:
    name: chrony
    state: latest

- name: config chronyd
  template:
    src: chrony.j2
    dest: /etc/chrony.conf
  notify:
  - restart chronyd
```

The chronyd configuration file is generated via the template module. A base template is supplied to Ansible and variable substituion is performed on it. This is a simple example, only using the *ntp_server* variable defined previously:

roles/common/templates/chrony.j2

```
server {{ ntp_server }} iburst
stratumweight 0
driftfile /var/lib/chrony/drift
rtcsync
makestep 10 3
```

```
bindcmdaddress 127.0.0.1
bindcmdaddress ::1
keyfile /etc/chrony.keys
commandkey 1
generatecommandkey
noclientlog
logchange 0.5
logdir /var/log/chrony
```

The *notify* argument is used to trigger a *handler*. A handler is a special type of task that is run after another task *notifies* that it is changed. In this case, the chronyd daemon is enabled and restarted when the configuration file is created:

roles/common/handlers/main.yaml

```
---
- name: restart chronyd
  service:
    name: chronyd
    state: restarted
    enabled: true
```

Finally, the role includes *motd.yaml* which is used to create a customized message of the day file:

roles/common/tasks/motd.yaml

```
---
- name: get build date
  stat:
    path: /root/anaconda-ks.cfg
  register: build_stat
- name: convert to a nice string for the template
  command: date --date='@{{ build_stat.stat.ctime }}' +'%Y.%m.%d @
%H:%M %Z'
  register: pretty_date
- name: create motd file
  template:
    src: motd.j2
    dest: /etc/motd
```

The *stat* module is used to get the details of a file created at build time, and converts the creation time in to a friendly format. That variable is then fed to another template, *motd.j2*:

roles/common/templates/motd.j2

```
welcome to {{ ansible_fqdn }} built on {{ pretty_date.stdout }}
```

hpe-predeploy

When deploying a complex infrastructure such as Red Hat OpenShift Container Platform, there are usually some prerequisite steps to prepare the environment, before installing the software itself.

The prerequisite tasks for this OpenShift reference architecture are captured in the *hp_predeploy* role.

playbooks/predeploy.yaml

```
---
- name: run predeployment tasks
  hosts: all
  gather_facts: true
  remote_user: root
  roles:
    - ../roles/passwords
    - ../roles/predeploy
  environment:
    PYTHONPATH: "$PYTHONPATH:/usr/share/ansible"
```

This role includes just three playbooks:

roles/predeploy/tasks/main.yaml

```
---
- include: dns.yaml
  tags:
  - dns
- include: packages.yaml
  tags:
  - packages
- include: docker.yaml
  tags:
  - docker
```

dns.yaml will setup the DNS entries for the publicly available *openshift-master* endpoint. It will also create the wildcard DNS entry, which will host the deployed apps in OpenShift. Dynamic DNS creation is done against the Red Hat Satellite server which runs an integrated *named* server.

roles/predeploy/tasks/dns.yaml

```
---
- name: get IP of load balancer
  set_fact:
    lb_ip: "{{ hostvars[groups['lb'][0]]['ansible_default_ipv4']
['address'] }}"
  run_once: true
  delegate_to: localhost

- name: get IP of DNS server via the load balancer
  set_fact:
    dns_server: "{{ hostvars[groups['lb'][0]]['ansible_dns']
['nameservers'].0 }}"
  run_once: true
  delegate_to: localhost

- name: create DNS entry for master
  nsupdate:
    state: present
    key_name: "rndc-key"
    key_secret: "{{ rndc_key }}"
```

```
    server: "{{ dns_server }}"
    zone: "{{ ansible_domain }}"
    record: "{{ openshift_master_cluster_public_hostname.split('.').0
}}"
    value: "{{ lb_ip }}"
  run_once: true
  delegate_to: localhost

- set_fact:
    region: "{{ openshift_node_labels.region }}"
  when:
  - openshift_node_labels is defined

- name: set subdomain record
  set_fact:
    subdomain: "{{ openshift_master_default_subdomain.split('.').0 }}"
  run_once: true
  delegate_to: localhost

- name: define router IP
  set_fact:
    router_ip: '{{ ansible_default_ipv4.address }}'
  delegate_to: localhost
  when:
  - region|default('none') == 'infra'

- name: blacklist non-router IPs
  set_fact:
    router_ip: '0.0.0.0'
  delegate_to: localhost
  when:
  - region|default('none') != 'infra'

- name: create router IP list
  set_fact:
    router_ips: "{{ groups['all']|map('extract', hostvars,
'router_ip')|list|unique|difference(['0.0.0.0']) }}"
  delegate_to: localhost
  run_once: true

- name: update wildcard DNS entry
  nsupdate:
    state: present
    key_name: "rndc-key"
    key_secret: "r/42yYsCOcnIxvXul+xz3Q=="
    server: "{{ dns_server }}"
    zone: "{{ ansible_domain }}"
    record: "*.{{ subdomain }}"
    value: "{{ router_ips }}"
  delegate_to: localhost
  run_once: true
```

This leans heavily on a modified *nsupdate* module. Currently the module does not support *one to many* record mappings. A pull request has been submitted upstream to handle this use case. The modified module has also been included in the git repo, under the *library* subdirectory.

packages.yaml ensures per-requisite packages for OpenShift are installed:

roles/predeploy/tasks/packages.yaml

```
---
- name: install ocp prereq packages
  package:
    name: '{{ item }}'
    state: latest
  with_items:
  - wget
  - git
  - net-tools
  - bind-utils
  - iptables-services
  - bridge-utils
  - bash-completion
  - kexec-tools
  - sos
  - psacct
```

And *docker.yaml* installs the docker RPMs and configures docker storage:

roles/predeploy/tasks/docker.yaml

```
---
- name: configure docker-storage-setup
  lineinfile:
    path: /etc/sysconfig/docker-storage-setup
    line: "VG=docker-vg"
    create: yes
  register: dss
```

hpe-cleanup

With an understanding of *hpe-provisioning*, *hpe-cleanup* merely does the opposite: to remove the servers from Satellite and HPE OneView. Although not necessary on an initial run, having such a playbook in a workflow is handy to facilitate a CI/CD (continuous integration/delivery) environment. When code is pushed to the git repo, or newer versions of the RPMs are available for testing, the entire workflow can be executed again, with the first step ensuring a clean build environment.

The *clean* playbook includes the *cleanup* role:

playbooks/clean.yaml

```
---
## clean up!

- name: get passwords
  hosts: all
  remote_user: root
  gather_facts: false
  roles:
    - ../roles/passwords
```

```
- name: delete nodes from satellite and oneview
  hosts: all
  remote_user: root
  gather_facts: false
  roles:
    - ../roles/cleanup
  environment:
    PYTHONPATH: "$PYTHONPATH:/usr/share/ansible"
    ONEVIEWSDK_IP: "{{ oneview_auth.ip }}"
    ONEVIEWSDK_USERNAME: "{{ oneview_auth.username }}"
    ONEVIEWSDK_PASSWORD: "{{ oneview_pw }}"
    ONEVIEWSDK_API_VERSION: "{{ oneview_auth.api_version }}"
```

Here's the directory structure of that role:

hpe-cleanup

```
├── roles
│   ├── cleanup
│   │   └── tasks
│   │       ├── dns.yaml
│   │       ├── facts.yaml
│   │       ├── main.yaml
│   │       ├── poweroff.yaml
│   │       ├── profiles.yaml
│   │       ├── rmserver.yaml
│   │       └── satellite.yaml
```

main.yaml is the entry point for the playbook and includes the remaining files in the proper order:

roles/cleanup/tasks/main.yaml

```
---
- include: ./facts.yaml
- include: ./poweroff.yaml
- include: ./satellite.yaml
- include: ./dns.yaml
- include: ./profiles.yaml
- include: ./rmserver.yaml
```

facts.yaml generates all the required variables while also checking to see if the server exists in HPE OneView or Red Hat Satellite:

roles/cleanup/tasks/facts.yaml

```
---
- set_fact:
    ilo_name: "{{ inventory_hostname.split('.').0 }}-ilo"
  tags:
  - ilo
- name: check if server is in oneview
  oneview_server_hardware_facts:
```

```
    name: "{{ ilo_name }}"
  register: server_facts_exists
  delegate_to: localhost
  tags:
  - mkserver
- set_fact:
    server_exists: false
  when:
  - server_facts_exists.ansible_facts.server_hardwares|length == 0
  tags:
  - mkserver
- set_fact:
    server_exists: true
  when:
  - server_facts_exists.ansible_facts.server_hardwares|length > 0
  tags:
  - mkserver
- set_fact:
    server_facts: "{{
server_facts_exists.ansible_facts.server_hardwares.0 }}"
  tags:
  - mkserver
  when:
  - server_exists == true

- set_fact:
      model: "{{ server_facts.model }}"
      short_model: "{{ server_facts.shortModel }}"
      dl_model: "{{ server_facts.shortModel.split(' ').0 }}"
  when:
  - server_exists == true

- name: get uri of profile template
  oneview_server_profile_template_facts:
    params:
      filter: name='OCP-{{ dl_model }}'
  register: profile_output
  delegate_to: localhost
  when:
  - server_exists == true

- set_fact:
    template_uri: "{{
profile_output.ansible_facts.server_profile_templates.0.uri }}"
  when:
  - server_exists == true
  - profile_output.ansible_facts.server_profile_templates|length > 0
```

poweroff.yaml shuts the node down:

roles/cleanup/tasks/poweroff.yaml

```
---
- name: power off server
  oneview_server_hardware:
```

```
    state: power_state_set
    data:
      name: "{{ ilo_name }}"
      powerStateData:
        powerState: "Off"
        powerControl: "PressAndHold"
  register: output
  delegate_to: localhost
  when:
  - server_exists == true
```

Then deleted in Satellite:

roles/cleanup/tasks/satellite.yaml

```
---
- name: "Get Host ID from Satellite"
  uri:
    url: "{{ satellite_url }}/api/hosts/?search=name={{
inventory_hostname }}"
    user: "{{ satellite_user }}"
    password: "{{ satellite_pw }}"
    headers:
      Content-Type: "application/json"
      Accept: "application/json"
    force_basic_auth: yes
    validate_certs: False
    return_content: yes
    status_code: 200
  ignore_errors: false
  register: check_host_response
  delegate_to: localhost

- name: set host ID
  set_fact:
    host_id: "{{ check_host_response.json.results.0.id }}"
  when:
  - check_host_response.json.subtotal == 1

- name: "delete host from satellite"
  uri:
    url: "{{ satellite_url }}/api/hosts/{{ host_id }}"
    user: "{{ satellite_user }}"
    password: "{{ satellite_pw }}"
    method: DELETE
    headers:
      Content-Type: "application/json"
      Accept: "application/json"
    force_basic_auth: yes
    validate_certs: False
    return_content: yes
    status_code: 200
  ignore_errors: false
```

```
  register: rm_host_response
  delegate_to: localhost
  when:
  - check_host_response.json.subtotal == 1
```

The OpenShift DNS entries are removed:

roles/cleanup/tasks/dns.yaml

```
---
- name: get IP of DNS server via the load balancer
  set_fact:
    dns_server: "{{ satellite_url.split('//').1 }}"
  run_once: true
  delegate_to: localhost

- name: set subdomain record
  set_fact:
    subdomain: "{{ openshift_master_default_subdomain.split('.').0 }}"
  run_once: true
  delegate_to: localhost

- name: remove DNS entry for master
  nsupdate:
    state: absent
    key_name: "rndc-key"
    key_secret: "{{ rndc_key }}"
    server: "{{ dns_server }}"
    zone: "{{ ansible_domain }}"
    record: "{{ openshift_master_cluster_public_hostname.split('.').0
}}"
  run_once: true
  delegate_to: localhost


- name: delete wildcard DNS entries
  nsupdate:
    state: absent
    key_name: "rndc-key"
    key_secret: "{{ rndc_key }}"
    server: "{{ dns_server }}"
    zone: "{{ ansible_domain }}"
    record: "*.{{ subdomain }}"
  run_once: true
  delegate_to: localhost
```

The profile is disassociated from the node and removed in HPE OneView:

roles/cleanup/tasks/profiles.yaml

```
---
- name: unassign server profile
  oneview_server_profile:
    state: absent
```

```
    data:
      name: "{{ inventory_hostname.split('.').0 }}"
      server_template: "OCP-{{ dl_model }}"
      server_hardware: "{{ ilo_name }}"
  register: output
  delegate_to: localhost
  when:
  - server_exists == true
```

And finally, delete the node from HPE OneView:

roles/cleanup/tasks/rmserver.yaml

```
---
- name: remove a server from oneview
  oneview_server_hardware:
    state: absent
    data:
      name: "{{ ilo_name }}"
      force: false
      licensingIntent: "OneView"
      configurationState: "Managed"
  register: output
  delegate_to: localhost
  tags:
  - mkserver
  when:
  - server_exists == true
```

hpe-openshift

The official Red Hat OpenShift Container Platform ansible playbooks were used for this reference architecture. Detailing these playbooks is out of scope for this document. The playbooks can be found on github and instructions on how to use them can be found here

The playbooks require no modification to work with Ansible Tower. As per the installation guide, a host file is created and a number of variables are defined. This is documented at the beginning of this chatper.

hpe-cns

With a properly running OpenShift Container Platform cluster, Red Hat Container-native storage can now be deployed. The following playbooks automate the tasks from the official documentation, found here.

The *cns* playbook calls the *cns* role:

playbooks/cns.yaml

```
---
- name: deploy container native storage w/ gluster
  hosts: all
  gather_facts: true
  remote_user: root
  roles:
```

```
    - ../roles/passwords
    - ../roles/cns
  environment:
    PYTHONPATH: "$PYTHONPATH:/usr/share/ansible"
```

The role's directory tree details the various files used:

```
├── roles
│   ├── cns
│   │   ├── tasks
│   │   │   ├── cnsdeploy.yaml
│   │   │   ├── disks.yaml
│   │   │   ├── heketitopo.yaml
│   │   │   ├── iptables.yaml
│   │   │   ├── main.yaml
│   │   │   ├── oc.yaml
│   │   │   └── packages.yaml
│   │   └── templates
│   │       └── topology-hpe.j2
```

The *cns* roles includes a number of required tasks:

roles/cns/tasks/main.yaml

```
---
- include: disks.yaml
  when:
  - "'cns' in group_names"
  tags:
  - disks

- include: iptables.yaml
  when:
  - "'cns' in group_names"
  tags:
  - iptables

- include: packages.yaml
  when:
  - "ansible_fqdn == groups['masters'][0]"

- include: oc.yaml
  when:
   - "ansible_fqdn == groups['masters'][0]"

- include: heketitopo.yaml
  when:
  - "ansible_fqdn == groups['masters'][0]"
  tags:
  - json

- include: cnsdeploy.yaml
```

```
  when:
  - "ansible_fqdn == groups['masters'][0]"
  tags:
  - cnsdeploy
```

Note that some of these tasks will only run on the nodes in the *cns* group, while others will only running on the first OpenShift master node, due to the 'when' statements.

disks.yaml will prepare the RAID volume for gluster: .roles/cns/tasks/disks.yaml

```
---
- name: nuke sdb on cns nodes for gluster
  command: sgdisk -Z /dev/sdb
```

gluster requires a number of firewall ports to be opened, which is handled by *iptables.yaml*:

roles/cns/tasks/iptables.yaml

```
---
- name: required gluster ports
  lineinfile:
    dest: /etc/sysconfig/iptables
    state: present
    insertbefore: 'COMMIT'
    line: "{{ item }}"
  with_items:
  - '-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
24007 -j ACCEPT'
  - '-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
24008 -j ACCEPT'
  - '-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport
2222 -j ACCEPT'
  - '-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m multiport --
dports 49152:49664 -j ACCEPT'

- name: reload iptables
  systemd:
    name: iptables
    state: reloaded
```

A couple of packages are required for CNS:

roles/cns/tasks/packages.yaml

```
---

- name: install cns and heketi packages
  package:
    name: '{{ item }}'
    state: latest
  with_items:
  - cns-deploy
  - heketi-client
```

The OpenShift environment is prepared by creating a project and setting some privileges:

roles/cns/tasks/oc.yaml

```
---
- name: loging as cns cluster admin
  command: oc login -u system:admin -n default

- name:  add cluster admin role for gluster
  command: oadm policy add-cluster-role-to-user cluster-admin gluster

- name: login as gluster user
  command: oc login -u gluster -p {{ gluster_pw }}

- name: create storage project
  command: oc new-project rhgs-cluster

- name: add scc policy default
  command: oadm policy add-scc-to-user privileged -z default

- name: add scc policy router
  command: oadm policy add-scc-to-user privileged -z router
```

heketi is used to install gluster and requires a json file describing the topology of the storage nodes. *heketitopo.yaml* generates this topology using the template module and facts from the hosts:

roles/cns/tasks/heketitopo.yaml

```
---
- template:
   src: topology-hpe.j2
   dest: /usr/share/heketi/topology-hpe.json
   owner: bin
   group: wheel
   mode: 0644
```

The template iterates over the *cns* group to define the hosts:

roles/cns/templates/topology-hpe.j2

```
{
  "clusters": [
    {
      "nodes": [
        {% for host in groups['cns'] %}
          {
          "node": {
            "hostnames": {
              "manage": [
                "{{ host }}"
              ],
              "storage": [
                "{{ hostvars[host]['ansible_default_ipv4']['address']
```

```
}}"
              ]
              },
              "zone": 1
              },
              "devices": [
                "/dev/sdb"
              ]
{% if loop.last %}
           }
{% else %}
          },
{% endif %}
{% endfor %}
        ]
    }
  ]
}
```

The generated file is then used to deploy the CNS solution:

roles/cns/tasks/cnsdeploy.yaml

```
---
- name: loging as cns gluster
  command: oc login -u gluster -p {{ gluster_pw }}


- name: deploy cns
  command: cns-deploy -n rhgs-cluster -g /usr/share/heketi/topology-
hpe.json -y  --admin-key {{ gluster_pw }} --user-key {{ gluster_pw }}
```

CHAPTER 7. DEPLOYING THE OPENSHIFT CONTAINER PLATFORM

This section provides the installation and configuration details for installing OpenShift Container Platform using Ansible Tower.

7.1. CLONE AND MODIFY GIT REPO

In order to deploy the playbooks created for this reference architecture, some modifications of the conrent are required. Notably, a password file encrypted with *ansible-vault* needs to be copied to *./roles/passwords/main.yaml* as outlined in the playbooks chapter. Environment specific changes should be made to the variables found under *./group_vars/*.

If an organization has an existing git infrastructure (such as gitlab) they may chose to clone the public repository and maintain their changes internally. Otherwise, the repository can be forked on github to track hold the requisite changes.

In either case, the modified playbooks must be stored in a git repository that is accessible to Ansible Tower.

7.2. HPE ONEVIEW MODULES

The HPE OneView Ansible modules are not a core module yet, so they are not included in a standard installation. They must be installed on the Ansible Tower server along with the required python packages.

First, install the HPE OneView SDK via **`pip`**:

```
curl https://bootstrap.pypa.io/get-pip.py
python get-pip.py
pip install hpOneView
```

Then clone the modules and make them accessible to Ansible Tower:

```
git clone https://github.com/HewlettPackard/oneview-ansible.git
cp oneview-ansible/library/* /usr/share/ansible/
```

7.3. MODIFIED NSUPDATE MODULE

As mentioned in the Playbooks chapter, the set of playbooks require a modified nsupdate module to support one to many DNS creation. A pull request has been submitted to support these feature. For now, the modified module must be copied to the Ansible Tower server:

```
curl -o /usr/share/ansible/nsupdate.py
https://raw.githubusercontent.com/dcritch/ansible/nsupdate_one2many/lib
/ansible/modules/net_tools/nsupdate.py
```

7.4. CREDENTIALS

Ansible typically communicates to a target host via SSH, using a public key. Ansible Tower can store and encrypt these credentials to run playbooks.

An SSH key pair is generated:

```
ssh-keygen -f tower
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in tower.
Your public key has been saved in tower.pub.
The key fingerprint is:
...
```

And then stored in Ansible Tower by clicking the **gear** icon, then **credentials** and finally **+ADD**:

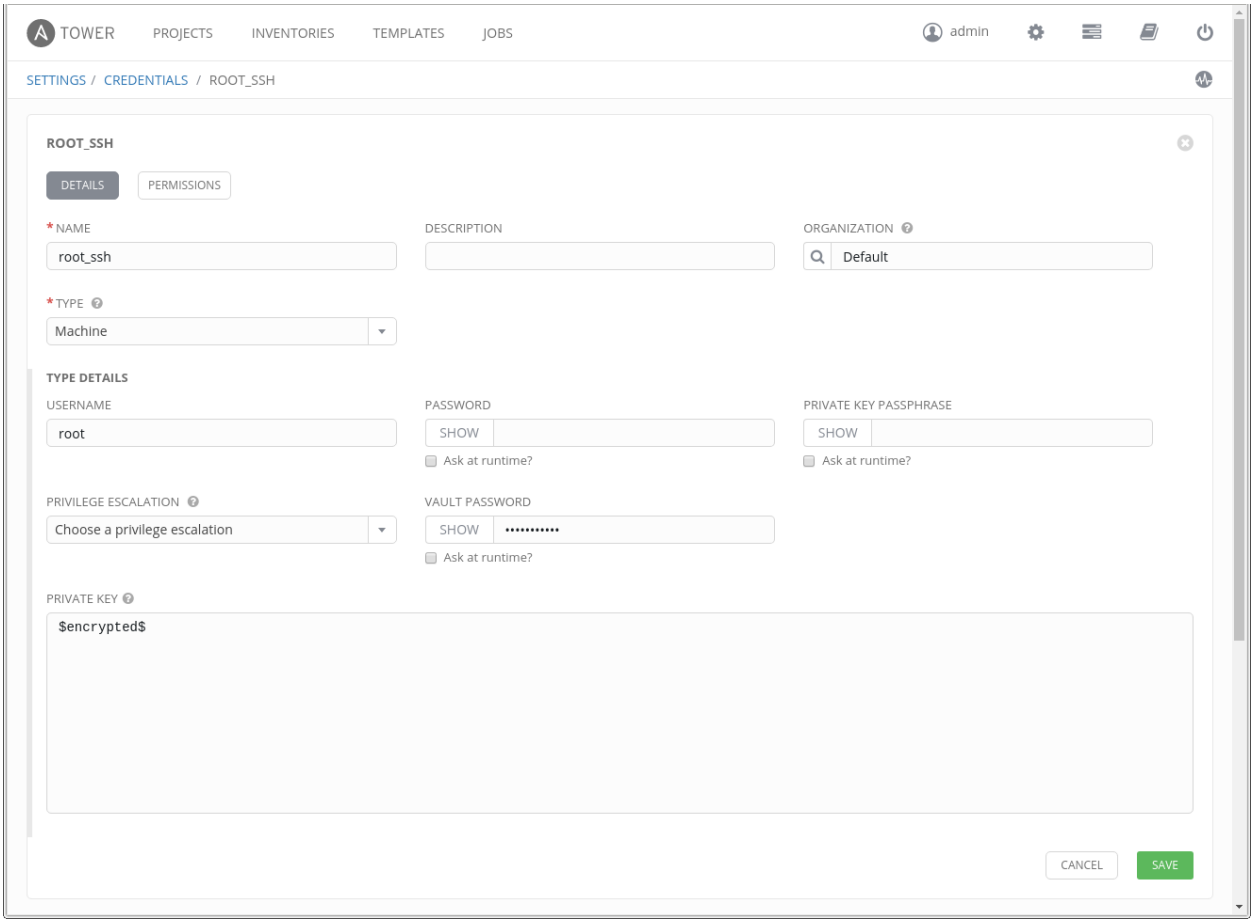


Figure 17 SSH credentials

The password used to encrypt the password playbook is also stored under 'Vault Password'.

The public SSH key is deployed via Red Hat Satellite when a host is provisioned.

In some cases, the git repository may require login details, which can also be stored safely in Ansible Tower:

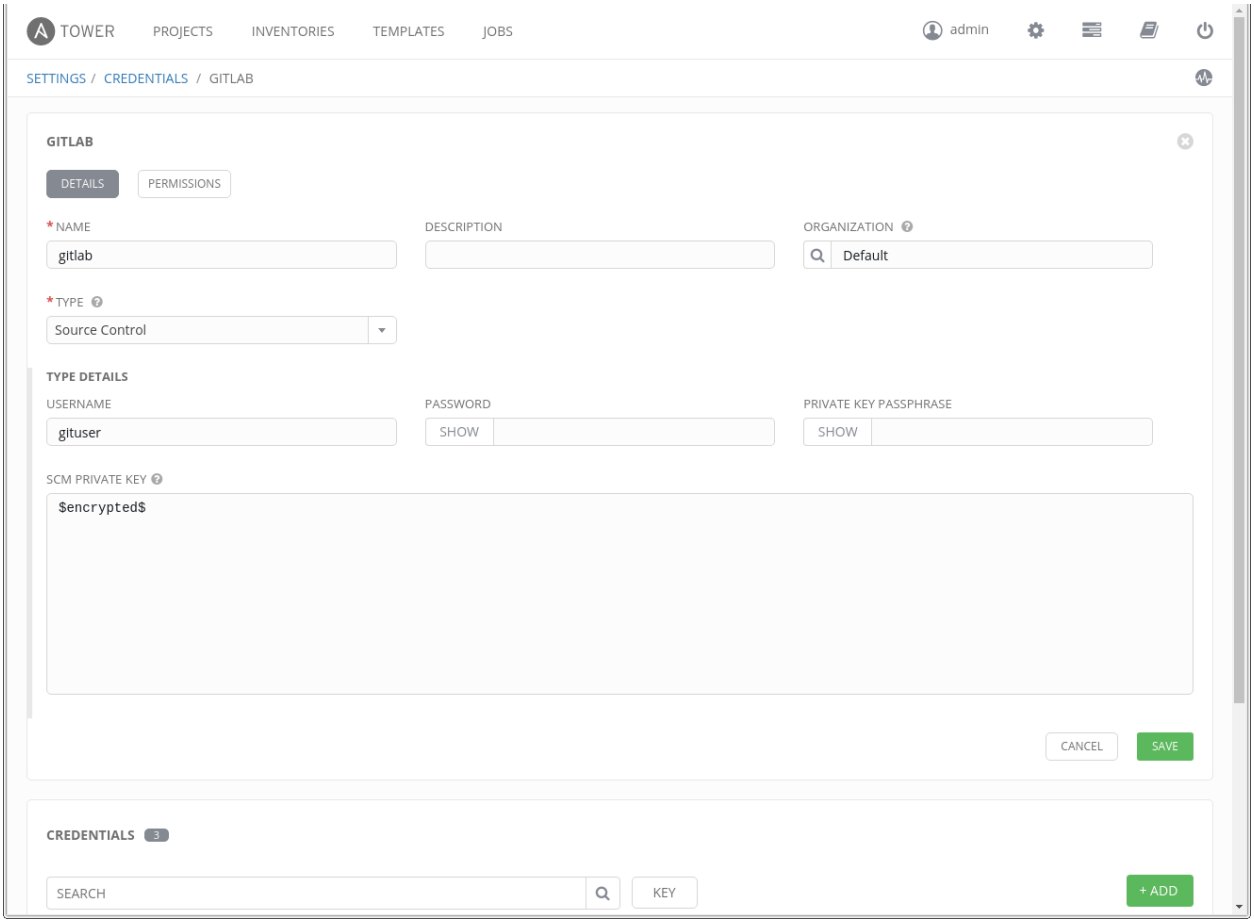


Figure 18 git credentials

7.5. PROJECTS

The locally maintained git repository is added to Ansible Tower:

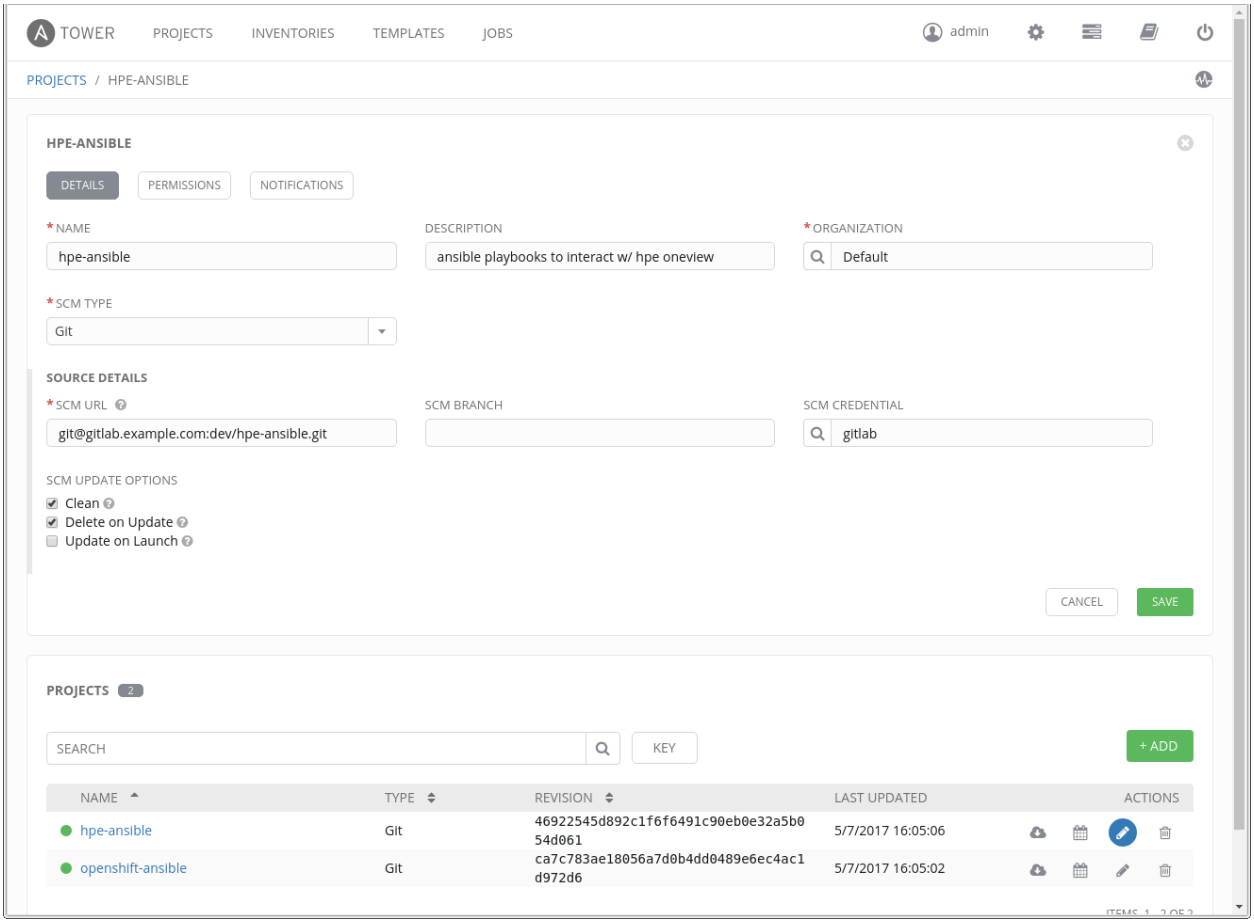


Figure 19 HPE Project

As well as the publically accessibly *openshift-ansible* playbooks:

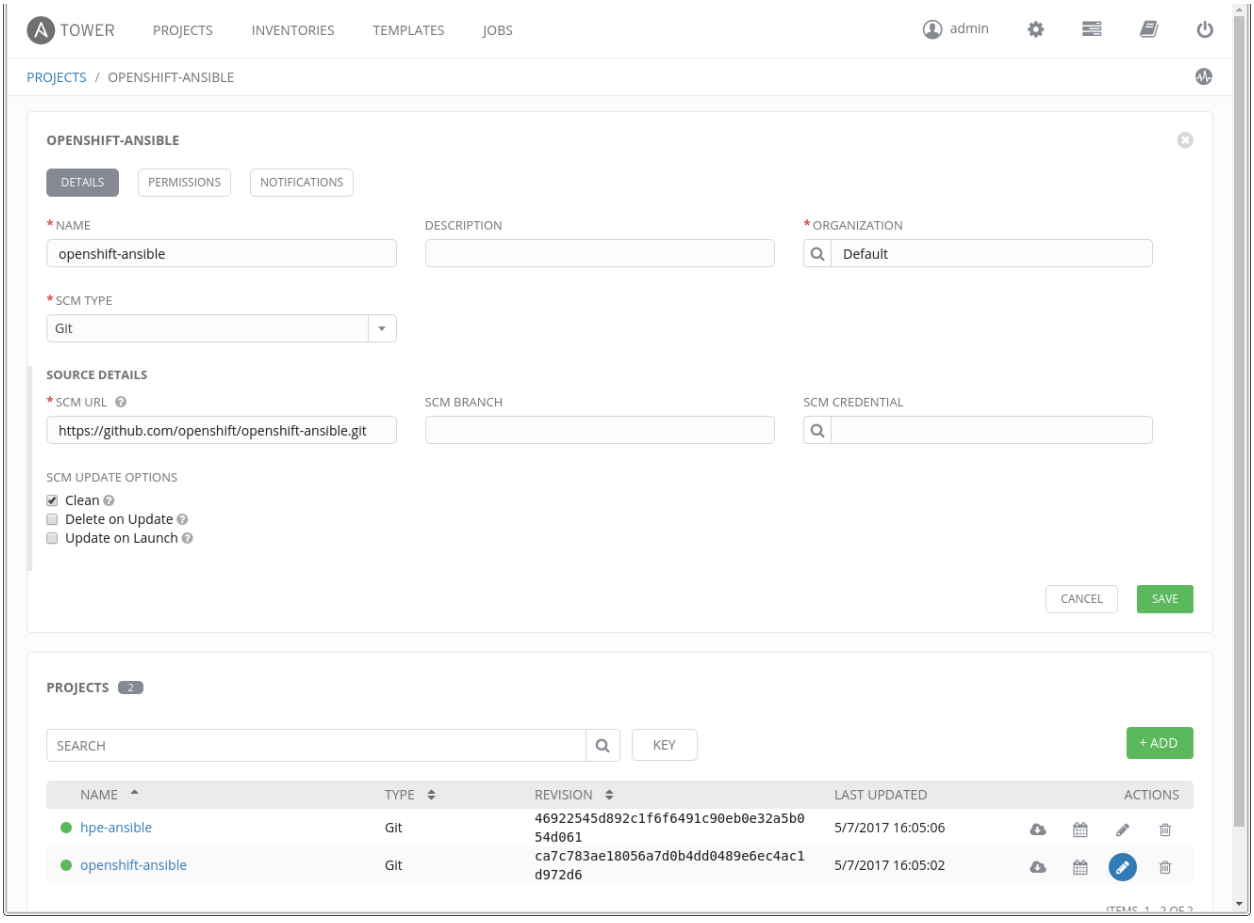


Figure 20 OpenShift Project

7.6. INVENTORIES

The list of hosts and variables for this reference architecture can be entered via the web GUI, however for complex playbooks with a lot of variables, this can be cumbersome. Ansible Tower provides a handy CLI tool that can be used to import this data.

Before using the tool, the empty inventory must first be created in the web GUI:

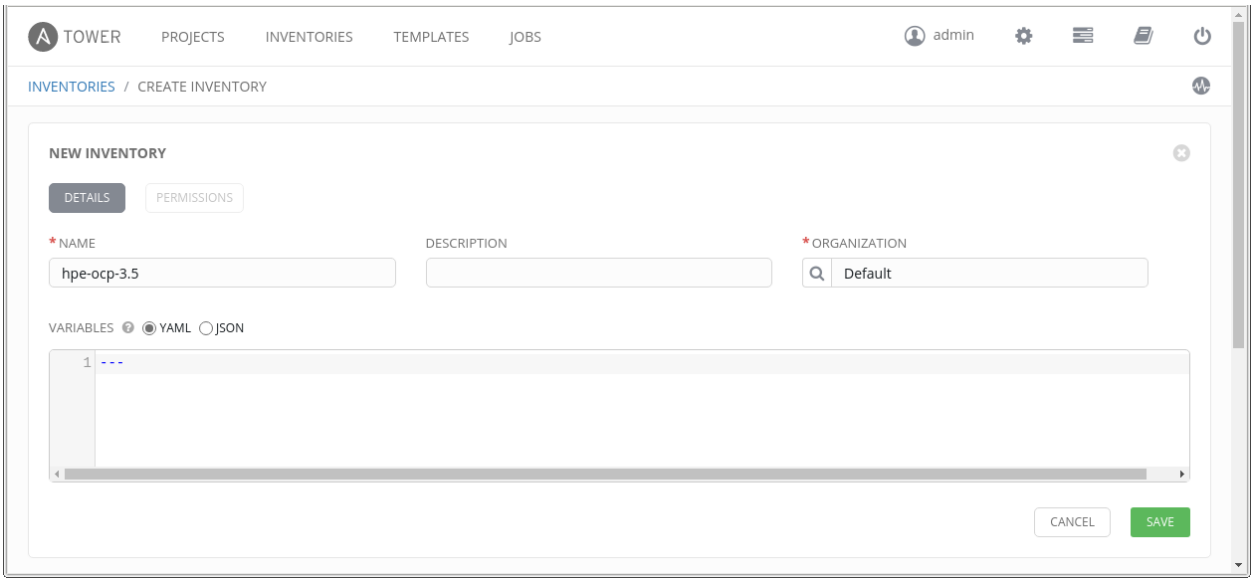


Figure 21 Ansible Tower Inventory

The host file and variables are copied to a directory on the Tower server:

```
ocp
├── ansible-hosts
└── group_vars
    ├── all
    ├── cns
    └── OSEv3
```

And then imported using **`tower-manage`**:

```
tower-manage inventory_import --source=./ocp/ --inventory-name="hpe-
ocp-3.5" --overwrite --overwrite-vars
```

Unfortunately there is one issue with this approach. Due to a bug, Tower can not parse certain inventory lines, for example:

```
ocp-cns1.hpecloud.test openshift_node_labels="{'region': 'primary',
'zone': 'east'}"
```

openshift_node_labels is interpreted as a string rather than a dictionary. After importing the inventory, the variable must be formatted slightly different to parse correctly:

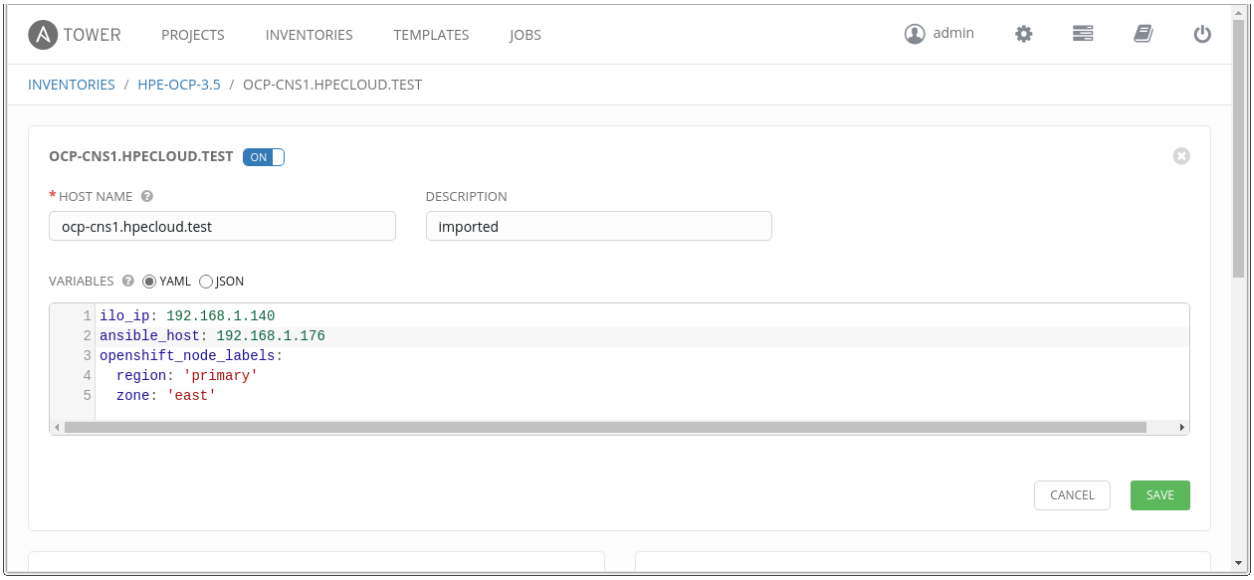


Figure 22 OpenShift Node Labels

7.7. TEMPLATES

With credentials, projects and inventories in place, the job templates can be created in Ansible Tower. This is accomplished by clicking on **Templates**, then **Add → Job Template** in Ansible Tower:

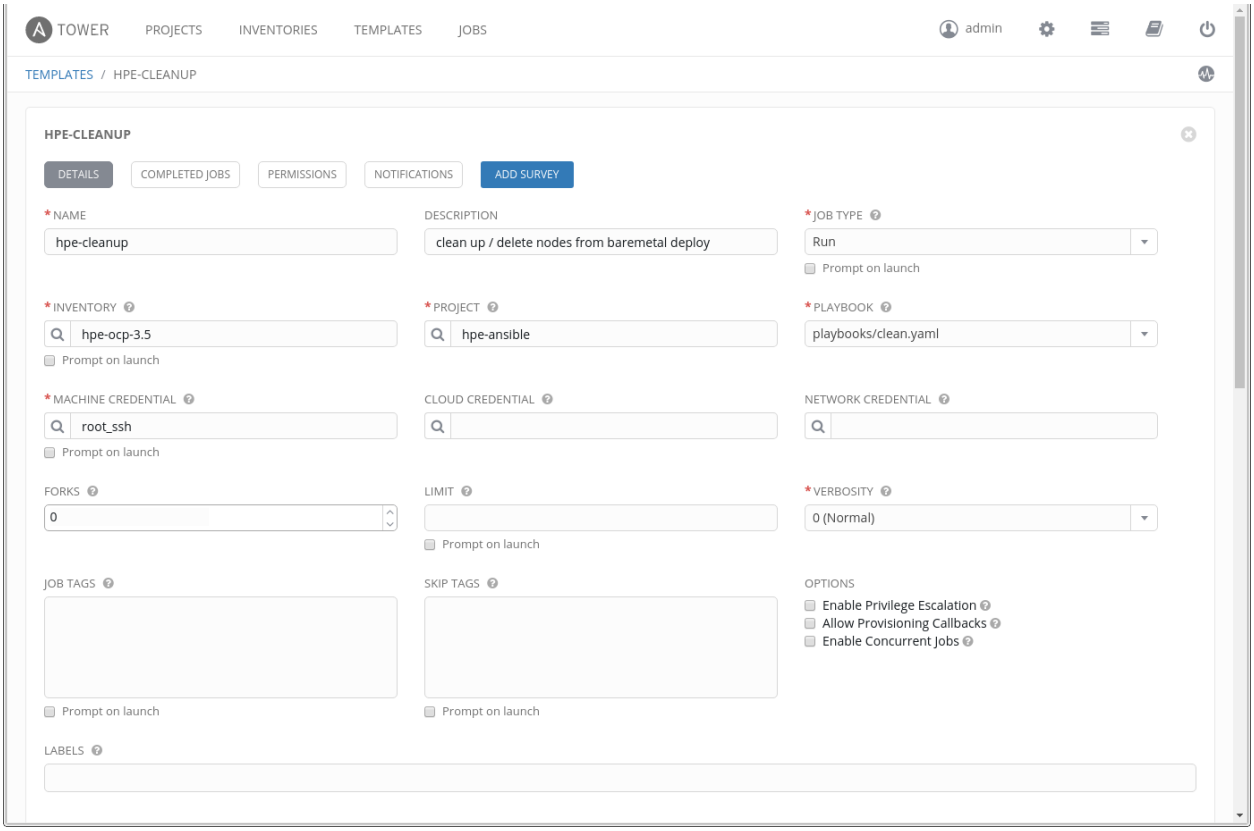


Figure 23 Cleanup Template

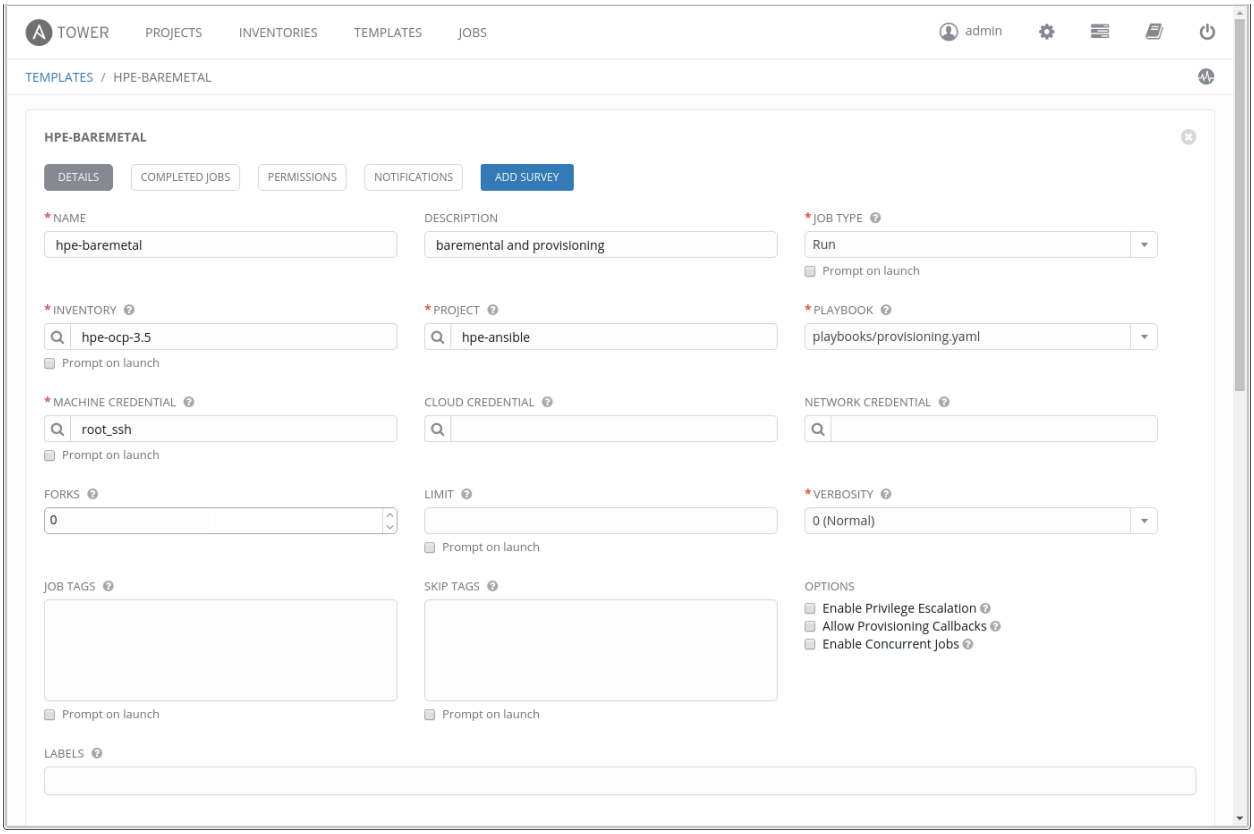


Figure 24 Provisioning Template

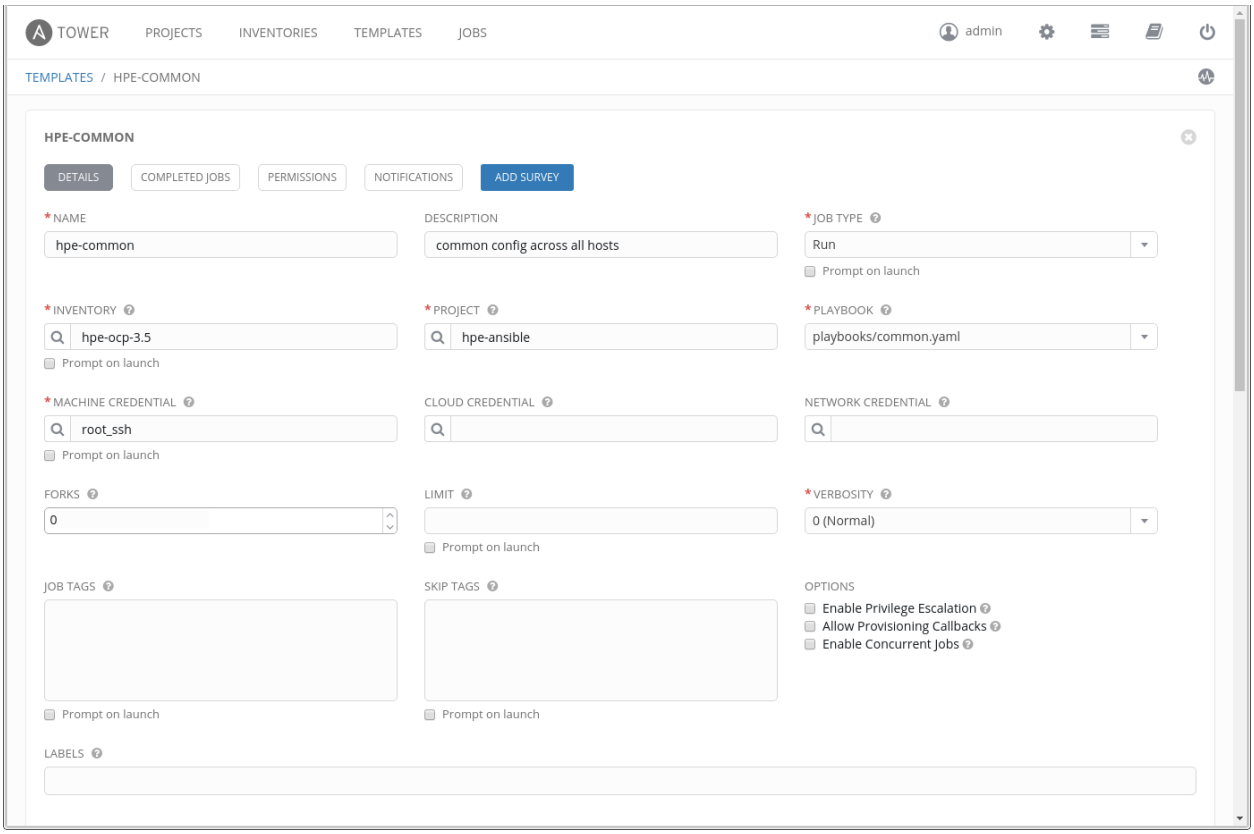


Figure 25 common Template

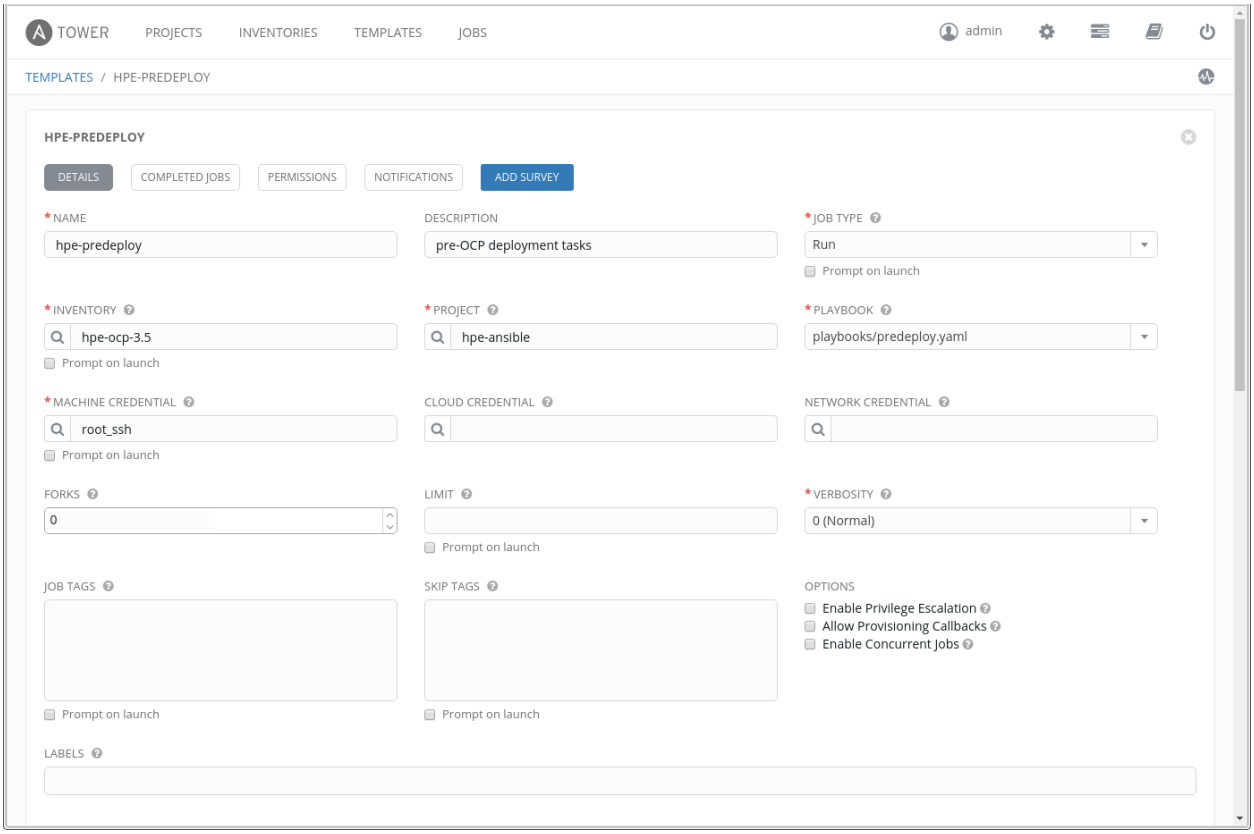


Figure 26 Cleanup Template

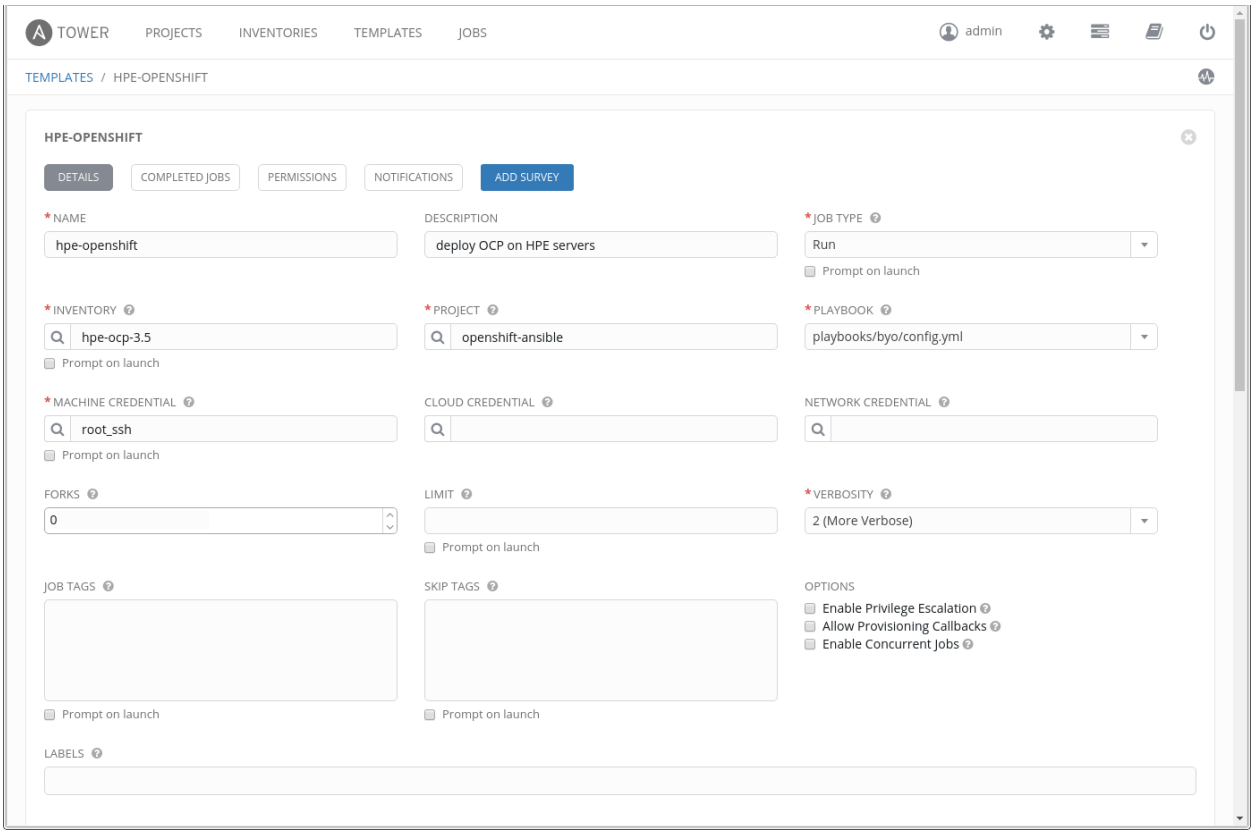


Figure 27 OpenShift Template

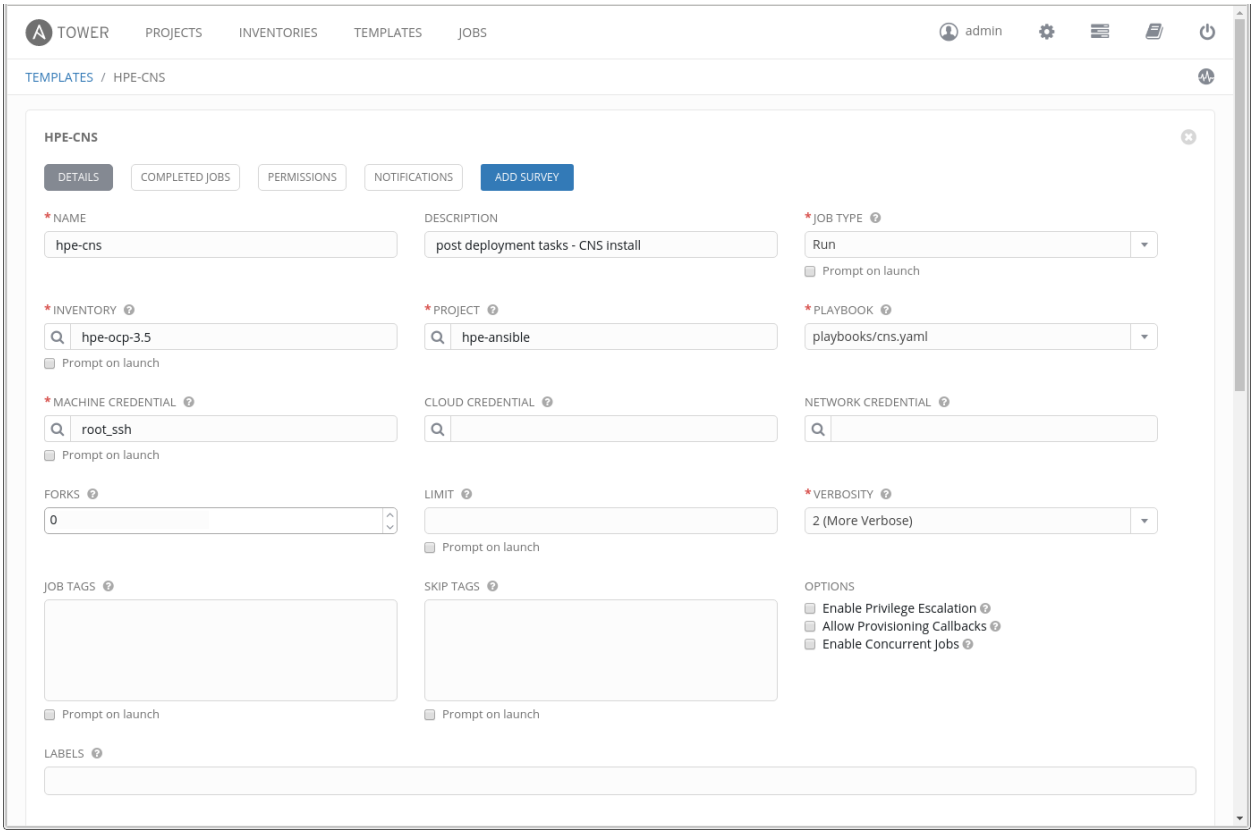


Figure 28 Container-native storage Template

7.8. WORKFLOW

The workflow to chain all the job templates together is then created by navigating to **Templates**, then **Add → Workflow Job Template**:

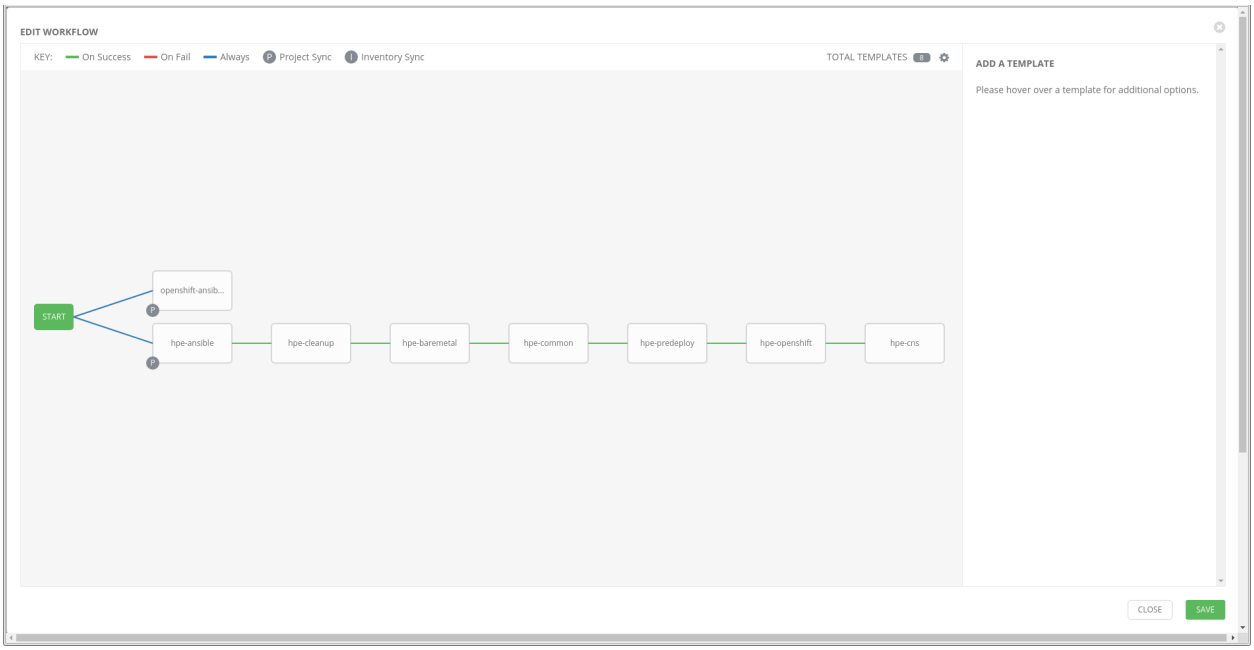


Figure 29 Ansible Tower Workflow

7.9. LAUNCH JOB

To launch the workflow, browse to **Templates**, find the newly created workflow and hit the rocket ship button:

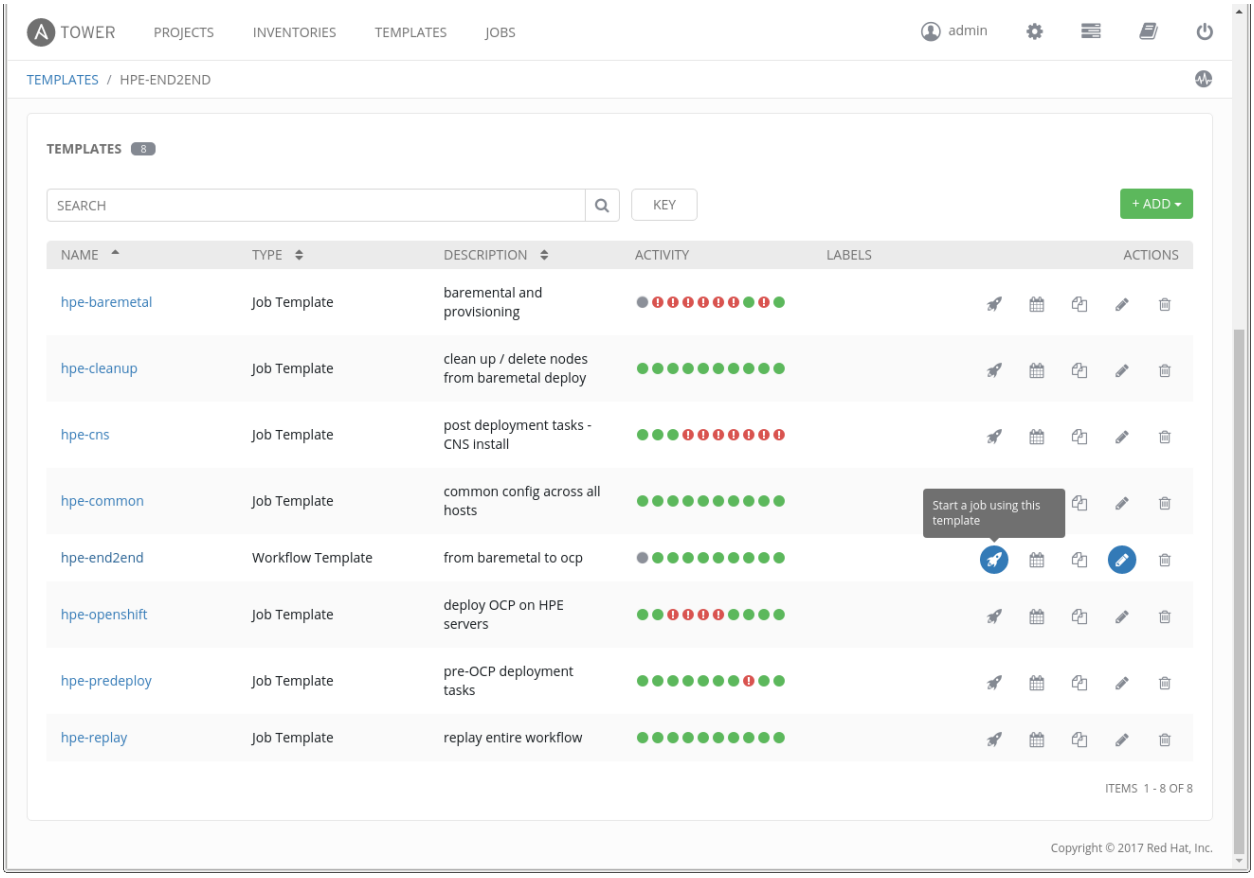


Figure 30 Launch Tower Workflow

CHAPTER 8. POST DEPLOYMENT AND VALIDATION

This section provides post deployment and validation information

Validate the Deployment Once the deployment is complete the Red Hat OpenShift Container Platform will resemble the deployment illustrated in the image below. The deployment will have three master nodes running the etcd cluster, three infrastructure nodes, and three Container-native storage Nodes. The registry for container applications is running on two of the infrastructure nodes. Moving the registry to persistent storage backed by Gluster Container-native storage is described later in this document. One of the infrastructure nodes is running HA Proxy to provide loadbalancing of external traffic among the master nodes. Routing services are running on two of the infrastructure nodes as well, the routers provide routing services between the external network and the container network. There are three nodes running Container-native storage to provide containers with persistent storage.

Figure 31 Red Hat OpenShift Container Platform Deployment

Verify the Red Hat OpenShift Platform ETCD cluster is running

To verify the Red Hat OpenShift Platfrom deployment was successful check the ectd cluster by logging into the first master node and running the following command:

```
etcdctl -C https://ocp-master1.hpecloud.test:2379,https://ocp-
master2.hpecloud.test:2379,https://ocp-master3.hpecloud.test:2379\
     --ca-file=/etc/origin/master/master.etcd-ca.crt \
     --cert-file=/etc/origin/master/master.etcd-client.crt \
     --key-file=/etc/origin/master/master.etcd-client.key cluster-health
2017-06-15 14:44:06.566247 I | warning: ignoring ServerName for user-
provided CA for backwards compatibility is deprecated
2017-06-15 14:44:06.566839 I | warning: ignoring ServerName for user-
provided CA for backwards compatibility is deprecated
member 70f2422ba978ec65 is healthy: got healthy result from
https://10.19.20.173:2379
member b6d844383be04fb2 is healthy: got healthy result from
https://10.19.20.175:2379
member fe30d0c37c03d494 is healthy: got healthy result from
https://10.19.20.174:2379
cluster is healthy
```

Check the members of the etcd cluster by executing the command *etcdctl -C* command as shown below:

```
etcdctl -C https://ocp-master1.hpecloud.test:2379,https://ocp-
master2.hpecloud.test:2379,https://ocp-master3.hpecloud.test:2379 \
```

```
     --ca-file=/etc/origin/master/master.etcd-ca.crt \
     --cert-file=/etc/origin/master/master.etcd-client.crt \
     --key-file=/etc/origin/master/master.etcd-client.key member list
2017-06-15 14:48:58.657439 I | warning: ignoring ServerName for user-
provided CA for backwards compatibility is deprecated
70f2422ba978ec65: name=ocp-master1.hpecloud.test
peerURLs=https://10.19.20.173:2380 clientURLs=https://10.19.20.173:2379
isLeader=false
b6d844383be04fb2: name=ocp-master3.hpecloud.test
peerURLs=https://10.19.20.175:2380 clientURLs=https://10.19.20.175:2379
isLeader=false
fe30d0c37c03d494: name=ocp-master2.hpecloud.test
peerURLs=https://10.19.20.174:2380 clientURLs=https://10.19.20.174:2379
isLeader=true
```

List nodes

Executing the *oc get nodes* command will display the OpenShift Container Platform nodes and their respective status.

```
oc get nodes
NAME                        STATUS                     AGE
ocp-cns1.hpecloud.test      Ready                      1h
ocp-cns2.hpecloud.test      Ready                      1h
ocp-cns3.hpecloud.test      Ready                      1h
ocp-infra1.hpecloud.test    Ready                      1h
ocp-infra2.hpecloud.test    Ready                      1h
ocp-master1.hpecloud.test   Ready,SchedulingDisabled   1h
ocp-master2.hpecloud.test   Ready,SchedulingDisabled   1h
ocp-master3.hpecloud.test   Ready,SchedulingDisabled   1h
```

List projects

Using the *oc get projects* command will display all projects that the user has permission to access.

```
oc get projects
NAME               DISPLAY NAME   STATUS
default                           Active
kube-system                       Active
logging                           Active
management-infra                  Active
openshift                         Active
openshift-infra                   Active
Using project "default" on server "https://openshift-
master.hpecloud.test:8443".
```

List pods

The *oc get pods* command will list the running pods.

```
oc get pods
NAME                        READY     STATUS    RESTARTS   AGE
docker-registry-1-sx2sh     1/1       Running   0          1h
docker-registry-1-tbz80     1/1       Running   0          1h
```

```
registry-console-1-sxm4d   1/1       Running   0          1h
router-1-ntql2             1/1       Running   0          1h
router-1-s6xqt             1/1       Running   0          1h
```

List Services

The *oc get services* command will list the available services, Cluster IP addresses and ports.

```
oc get services
NAME               CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
docker-registry    172.30.80.1     <none>        5000/TCP
1h
kubernetes         172.30.0.1      <none>        443/TCP,53/UDP,53/TCP
1h
registry-console   172.30.105.3    <none>        9000/TCP
1h
router             172.30.206.66   <none>        80/TCP,443/TCP,1936/TCP
1h
```

List endpoints

The *oc get endpoints* command will display a list of external endpoint IP addresses and ports.

```
oc get endpoints
NAME               ENDPOINTS
AGE
docker-registry    10.128.2.4:5000,10.128.2.5:5000
1h
kubernetes         10.19.20.173:8443,10.19.20.174:8443,10.19.20.175:8443
+ 6 more...   1h
registry-console   10.128.2.3:9090
1h
router             10.19.20.170:443,10.19.20.171:443,10.19.20.170:1936 +
3 more...     1h
```

Verify the Red Hat OpenShift Container Platform User Interface

Log into the Red Hat OpenShift Container Platform user interface using the public url defined in the openshift_master_cluster_public_hostname variable as shown below.

openshift_master_cluster_public_hostname: openshift-master.hpecloud.test

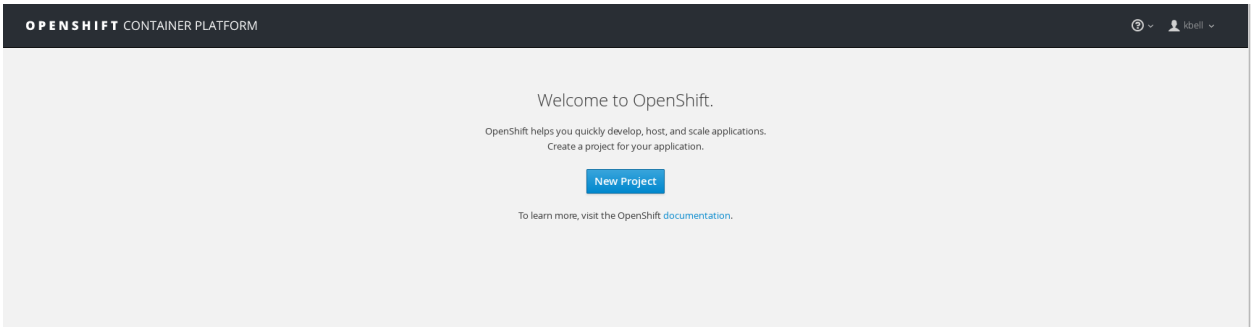


Figure 32 Red Hat OpenShift Container Platform User Interface

Verify Container-native storage

Container-native storage is a hyper-converged solution that allows containerized applications and Gluster based storage to reside on the same nodes. The Container-native storage provides containers with persistent storage. Secrets are defined to enhance security for Container-native storage, these secrets are applied to the configuration as variables in the cnsdeploy.yaml ansible task. The secrets are stored in an encrypted format in ansible vault and initially defined in the passwords.yaml file. There is an admin secret and a user secret defined. Set the environment variables by exporting the heketi user and the heketi secret to be used with the heketi cli command as HEKETI_CLI_USER and HEKETI_CLI_KEY, for example *export HEKETI_CLI_USER=admin* and *export HEKETI_CLI_KEY=n0tP@ssw0rd* . Alternatively, the user and secret can be passed on the command line using the --secret and --user options as shown below.

heketi-cli topology info -s http://heketi-rhgs-cluster.paas.hpecloud.test --user admin --secret n0tP@ssw0rd

Use the *oc get route heketi* command to get the heketi server name.

```
oc get route heketi
NAME      HOST/PORT                                   PATH      SERVICES
PORT      TERMINATION   WILDCARD
heketi    heketi-rhgs-cluster.paas.hpecloud.test              heketi
<all>                   None
```

Export the heketi server name as HEKETI_CLI_SERVER by executing *export HEKETI_CLI_SERVER=http://heketi-rhgs-cluster.paas.hpecloud.test* .

Using the heketi client create a 100 GB test volume on the cluster named test_volume

```
heketi-cli volume create --size=100 --name test_volume
Name: test_volume
Size: 100
Volume Id: b39a544984eb9848a1727564ea66cb6f
Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
Mount: 10.19.20.176:test_volume
Mount Options: backup-volfile-servers=10.19.20.177,10.19.20.178
Durability Type: replicate
Distributed+Replica: 3
```

In our example the heketi server name is heketi-rhgs-cluster.paas.hpecloud.test, export the server name the to the HEKETI_CLI_SERVER variable. Use the heketi client, *heketi-cli* , to view the Gluster topology as shown below. Notice the 100 GB volume previously created test_volume. To delete the test volume use *heketi-cli volume delete <volume ID>* .

```
heketi-cli topology info
Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
    Volumes:
 Name: heketidbstorage
 Size: 2
 Id: 51b91b87d77bff02cb5670cbab621d6d
 Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
 Mount: 10.19.20.176:heketidbstorage
 Mount Options: backup-volfile-servers=10.19.20.177,10.19.20.178
 Durability Type: replicate
 Replica: 3
 Snapshot: Disabled
  Bricks:
   Id: 7f243adb17ab1dbf64f032bff7c47d81
```

```
   Path:
/var/lib/heketi/mounts/vg_5e25001fb558bff388f3304897440f33/brick_7f243adb
17ab1dbf64f032bff7c47d81/brick
   Size (GiB): 2
   Node: 0fd823a6c17500ffcfdf797f9eaa9d4a
   Device: 5e25001fb558bff388f3304897440f33
-
   Id: 8219d3b014d751bbb58fcfae57eb1cf0
   Path:
/var/lib/heketi/mounts/vg_159c4b0cca850b7549b3957e93014ffe/brick_8219d3b0
14d751bbb58fcfae57eb1cf0/brick
   Size (GiB): 2
   Node: 8b1e2abf4076ef4dd93299f2be042b89
   Device: 159c4b0cca850b7549b3957e93014ffe
-
   Id: 843b35abc2ab505ed7591a2873cdd29d
   Path:
/var/lib/heketi/mounts/vg_01423a48fb095139ce7b229a73eac064/brick_843b35ab
c2ab505ed7591a2873cdd29d/brick
   Size (GiB): 2
   Node: a5923b6b35fef3131bd077a33f658239
   Device: 01423a48fb095139ce7b229a73eac064
 Name: test_volume
 Size: 100
 Id: b39a544984eb9848a1727564ea66cb6f
 Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
 Mount: 10.19.20.176:test_volume
 Mount Options: backup-volfile-servers=10.19.20.177,10.19.20.178
 Durability Type: replicate
 Replica: 3
 Snapshot: Disabled
  Bricks:
   Id: 45cc45e08b6215ee206093489fa12216
   Path:
/var/lib/heketi/mounts/vg_01423a48fb095139ce7b229a73eac064/brick_45cc45e0
8b6215ee206093489fa12216/brick
   Size (GiB): 100
   Node: a5923b6b35fef3131bd077a33f658239
   Device: 01423a48fb095139ce7b229a73eac064
-
   Id: 47722ac826e6a263c842fe6a87f30d84
   Path:
/var/lib/heketi/mounts/vg_159c4b0cca850b7549b3957e93014ffe/brick_47722ac8
26e6a263c842fe6a87f30d84/brick
   Size (GiB): 100
   Node: 8b1e2abf4076ef4dd93299f2be042b89
   Device: 159c4b0cca850b7549b3957e93014ffe
-
   Id: a257cdad1725fcfa22de954e59e70b64
   Path:
/var/lib/heketi/mounts/vg_5e25001fb558bff388f3304897440f33/brick_a257cdad
1725fcfa22de954e59e70b64/brick
   Size (GiB): 100
   Node: 0fd823a6c17500ffcfdf797f9eaa9d4a
   Device: 5e25001fb558bff388f3304897440f33
    Nodes:
```

```
 Node Id: 0fd823a6c17500ffcfdf797f9eaa9d4a
 State: online
 Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
 Zone: 1
 Management Hostname: ocp-cns3.hpecloud.test
 Storage Hostname: 10.19.20.178
 Devices:
  Id:5e25001fb558bff388f3304897440f33   Name:/dev/sdb
State:online    Size (GiB):11177   Used (GiB):102     Free (GiB):11075
   Bricks:
    Id:7f243adb17ab1dbf64f032bff7c47d81   Size (GiB):2       Path:
/var/lib/heketi/mounts/vg_5e25001fb558bff388f3304897440f33/brick_7f243adb
17ab1dbf64f032bff7c47d81/brick
    Id:a257cdad1725fcfa22de954e59e70b64   Size (GiB):100     Path:
/var/lib/heketi/mounts/vg_5e25001fb558bff388f3304897440f33/brick_a257cdad
1725fcfa22de954e59e70b64/brick
 Node Id: 8b1e2abf4076ef4dd93299f2be042b89
 State: online
 Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
 Zone: 1
 Management Hostname: ocp-cns2.hpecloud.test
 Storage Hostname: 10.19.20.177
 Devices:
  Id:159c4b0cca850b7549b3957e93014ffe   Name:/dev/sdb
State:online    Size (GiB):11177   Used (GiB):102     Free (GiB):11075
   Bricks:
    Id:47722ac826e6a263c842fe6a87f30d84   Size (GiB):100     Path:
/var/lib/heketi/mounts/vg_159c4b0cca850b7549b3957e93014ffe/brick_47722ac8
26e6a263c842fe6a87f30d84/brick
    Id:8219d3b014d751bbb58fcfae57eb1cf0   Size (GiB):2       Path:
/var/lib/heketi/mounts/vg_159c4b0cca850b7549b3957e93014ffe/brick_8219d3b0
14d751bbb58fcfae57eb1cf0/brick
 Node Id: a5923b6b35fef3131bd077a33f658239
 State: online
 Cluster Id: 05f9e98b350c2341c1dffb746d0549e9
 Zone: 1
 Management Hostname: ocp-cns1.hpecloud.test
 Storage Hostname: 10.19.20.176
 Devices:
  Id:01423a48fb095139ce7b229a73eac064   Name:/dev/sdb
State:online    Size (GiB):11177   Used (GiB):102     Free (GiB):11075
   Bricks:
    Id:45cc45e08b6215ee206093489fa12216   Size (GiB):100     Path:
/var/lib/heketi/mounts/vg_01423a48fb095139ce7b229a73eac064/brick_45cc45e0
8b6215ee206093489fa12216/brick
    Id:843b35abc2ab505ed7591a2873cdd29d   Size (GiB):2       Path:
/var/lib/heketi/mounts/vg_01423a48fb095139ce7b229a73eac064/brick_843b35ab
c2ab505ed7591a2873cdd29d/brick
```

Backing the Docker Registry with Gluster Container-native storage

In a production environment the local Docker registry should be backed with persistent storage. This section will describe how to create a Persistent Volume Claim (PVC) and Persistent Volume (PV) for the Docker registry.

In this example a user account named gluster was created and assigned the cluster admin role. Log in under this account to perform the *oc* commands described in this section.

The following files are required to create the gluster service, endpoints, persistent volume and persistent volume claim. In this example the files were created on the master nodes, ocp-master1.hpecloud.test, and the *oc* commands were executed from this node.

Gluster Service File

The contents of the glusterservice.yaml file are shown below. This file will create an OpenShift service named glusterfs-cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: glusterfs-cluster
spec:
  ports:
  - port: 1
```

Gluster Endpoint File

The contents of the glusterendpoints.yaml file are shown below, this file defines the gluster endpoints. The IP addresses in this file are the IP addresses of the Gluster nodes.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: glusterfs-cluster
subsets:
  - addresses:
      - ip: 10.19.20.176
    ports:
      - port: 1
  - addresses:
      - ip: 10.19.20.177
    ports:
      - port: 1
  - addresses:
      - ip: 10.19.20.178
    ports:
      - port: 1
```

Gluster PV File

The contents of the glusterpv.yaml file is shown below, this file describes the persistent volume that will be used for the Docker registry persistent storage. The path specified must match the name of the volume that will be used for the Docker registry on the Gluster Container-native storage.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-default-volume
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
```

```
    endpoints: glusterfs-cluster
    path: regVol1
    readOnly: false
  persistentVolumeReclaimPolicy: Retain
```

Gluster PVC File

The contents of the glusterpvc.yaml file are shown below. This file describes the persistent volume claim that will be used by the Docker registry pods to mount the persistent volume and provide the Docker registry with persistent storage.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gluster-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

Creating a volume for the Docker registry

Use the *heketi-cli volume create* command to create a volume on the Container-native storage for the Docker registry persistent storage. The name used when creating the volume must match the path specified in the glusterpv.yaml file.

```
heketi-cli volume create --size=20 --name=regVol1
```

The code snipet below shows the commands along with thier respective outputs that are used to create the glusterfs service, the gluster endpoints, the persistent volume and the persistent volume claim that will be used to provide persistent storage for the Docker registry.

```
[root@ocp-master1 ~]# oc whoami
gluster
[root@ocp-master1 ~]# oc create -f glusterservice.yaml
service "glusterfs-cluster" created
[root@ocp-master1 ~]# oc get services
NAME                CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
docker-registry     172.30.35.174    <none>        5000/TCP
8h
glusterfs-cluster   172.30.81.99     <none>        1/TCP
5s
kubernetes          172.30.0.1       <none>        443/TCP,53/UDP,53/TCP
8h
registry-console    172.30.173.160   <none>        9000/TCP
8h
router              172.30.139.18    <none>
80/TCP,443/TCP,1936/TCP   8h
[root@ocp-master1 ~]# oc create -f glusterendpoints.yaml
endpoints "glusterfs-cluster" created
[root@ocp-master1 ~]# oc get endpoints
NAME                ENDPOINTS
AGE
```

```
docker-registry     10.128.2.3:5000,10.131.0.3:5000
8h
glusterfs-cluster   10.19.20.176:1,10.19.20.177:1,10.19.20.178:1
27s
kubernetes          10.19.20.173:8443,10.19.20.174:8443,10.19.20.175:8443
+ 6 more...   8h
registry-console    10.130.0.2:9090
8h
router              10.19.20.170:443,10.19.20.171:443,10.19.20.170:1936 +
3 more...     8h
[root@ocp-master1 ~]# oc create -f glusterpv.yaml
persistentvolume "gluster-default-volume" created
[root@ocp-master1 ~]# oc get pv
NAME                    CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM                   REASON    AGE
gluster-default-volume   20Gi       RWX           Retain
Available                                      16s
registry-volume          10Gi       RWX           Retain          Bound
default/registry-claim             8h
[root@ocp-master1 ~]# oc create -f glusterpvc.yaml
persistentvolumeclaim "gluster-claim" created
[root@ocp-master1 ~]# oc get pvc
NAME             STATUS    VOLUME                   CAPACITY
ACCESSMODES   AGE
gluster-claim    Bound     gluster-default-volume   20Gi       RWX
25s
registry-claim   Bound     registry-volume          10Gi       RWX
8h
```

Attach the Docker registry to the persistent storage using the *oc volume* command shown below.

```
oc volume deploymentconfigs/docker-registry --add --name=v1 -t pvc --
claim-name=gluster-claim --overwrite
```

Review the OpenShift documentation at https://docs.openshift.com/container-platform/3.5/install_config/storage_examples/gluster_backed_registry.html for detailed information on backing the Docker registry with Gluster File Storage.

CNS Storage Class

This section will describe how to create a storage class for provisioning storage from the gluster based Container-native storage. To create an OpenShift storage class you will need the cluster ID and the rest API URL of the CNS cluster. The rest API URL is the same as the HEKETI_CLI_SERVER variable exported earlier. Use the *heketi-cli cluster list* command to display the cluster ID and the *oc get route* command to display the heketi server name.

```
[root@ocp-master1 ~]# heketi-cli cluster list
Clusters:
5f64ac6c1313d2abba1143bb808da586
[root@ocp-master1 ~]# oc get route
NAME      HOST/PORT                                PATH      SERVICES
PORT      TERMINATION   WILDCARD
heketi    heketi-rhgs-cluster.paas.hpecloud.test             heketi
<all>                   None
[root@ocp-master1 ~]# echo $HEKETI_CLI_SERVER
http://heketi-rhgs-cluster.paas.hpecloud.test
```

The storage class object requires the heketi secret that was created during provisioning of the Container-native storage. In this example, the heketi secret is "n0tP@ssw0rd". This secret will be stored in OpenShift as a secret. The OpenShift secret is created using the *oc create* command along with a file that defines the secret. First we are going to encrypt the secret with the *echo -n <secret> | base64* command, then create a yaml file that will be passed with the*oc create* command to create the secret in OpenShift as shown below.

```
[root@ocp-master1 ~]# echo -n n0tP@ssw0rd |base64
bjB0UEBzc3cwcmQ=
[root@ocp-master1 ~]# vi heketi-secret.yaml
[root@ocp-master1 ~]# oc create -f heketi-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bjB0UEBzc3cwcmQ=
type: kubernetes.io/glusterfs
[root@ocp-master1 ~]# cat heketi-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bjB0UEBzc3cwcmQ=
type: kubernetes.io/glusterfs
[root@ocp-master1 ~]# oc create -f heketi-secret.yaml
secret "heketi-secret" created
[root@ocp-master1 ~]# oc describe secret heketi-secret
Name:  heketi-secret
Namespace: default
Labels:  <none>
Annotations: <none>
Type: kubernetes.io/glusterfs
Data
====
key: 11 bytes
```

Next, create a yaml file that will be used to create the storageclass object. This file will contain the cns cluster id, cluster name, and the heketi username and secret.

```
[root@ocp-master1 ~]# vi cns-storageclass-st1.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-bronze
provisioner: kubernetes.io/glusterfs
parameters:
  clusterid: 5f64ac6c1313d2abba1143bb808da586
  resturl: http://heketi-rhgs-cluster.paas.hpecloud.test
  restauthenabled: "true"
  restuser: "admin"
  secretName: "heketi-secret"
  secretNamespace: "default"
```

The file will be passed with the *oc create* command to create a new storage class object named gluster-cns-bronze.

```
[root@ocp-master1 ~]# oc create -f cns-storageclass-st1.yaml
storageclass "gluster-cns-bronze" created
[root@ocp-master1 ~]# oc describe storageclass gluster-cns-bronze
Name:  gluster-cns-bronze
IsDefaultClass: No
Annotations: <none>
Provisioner: kubernetes.io/glusterfs
Parameters:
clusterid=5f64ac6c1313d2abba1143bb808da586,restauthenabled=true,resturl=h
ttp://heketi-rhgs-
cluster.paas.hpecloud.test,restuser=admin,secretName=n0tP@ssw0rd,secretNa
mespace=default
No events.
```

Now that the storage class object, gluster-cns-bronze, has been created, create a persistent volume claim against the gluster-cns-bronze storage class. This persistent volume claim will dynamically allocate storage from the gluster-cns-bronze storage class.

Create a yaml file that defines the persistent volume claime against the gluster-cns-bronze storage class object as shown below.

```
[root@ocp-master1 ~]# vi cns-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-cns-bronze
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
```

Create the persistent volume claim using the *oc create* command and passing the cns-claim.yaml file.

```
[root@ocp-master1 ~]# oc create -f cns-claim.yaml
persistentvolumeclaim "db" created
```

Review the provisioning of the persistent storage using the *oc describe persistentvolumeclaim* command and *heketi-cli volume info* command

```
[root@ocp-master1 ~]# oc describe persistentvolumeclaim db
Name:  db
Namespace: rhgs-cluster
StorageClass: gluster-cns-bronze
Status:  Bound
Volume:  pvc-7605ae4b-714b-11e7-a319-1402ec825eec
Labels:  <none>
Capacity: 10Gi
```

```
Access Modes: RWO
No events.
[root@ocp-master1 ~]# heketi-cli volume info
93b150f2f7a5ddcb1ed5462cedeb43a9
Name: vol_93b150f2f7a5ddcb1ed5462cedeb43a9
Size: 10
Volume Id: 93b150f2f7a5ddcb1ed5462cedeb43a9
Cluster Id: 5f64ac6c1313d2abba1143bb808da586
Mount: 10.19.20.176:vol_93b150f2f7a5ddcb1ed5462cedeb43a9
Mount Options: backup-volfile-servers=10.19.20.177,10.19.20.178
Durability Type: replicate
Distributed+Replica: 3
```

CHAPTER 9. DEPLOY AN APPLICATION

In this reference architecture the cakephp application is used as a sample application. This sample application can be found at https://hub.openshift.com/quickstarts/73-cakephp. A list of example applications can be found at https://hub.openshift.com/quickstarts/all or in the OpenShift documentation at https://docs.openshift.com/container-platform/3.5/dev_guide/migrating_applications/quickstart_examples.html.

Access the pubic url endpoint of the OpenShift deployment at https://openshift-master.hpecloud.test:8443/console/command-line for instructions on setting up the command line interface. This link will display instructions on downloading and installing the oc commandline tools as shown in figure 33.

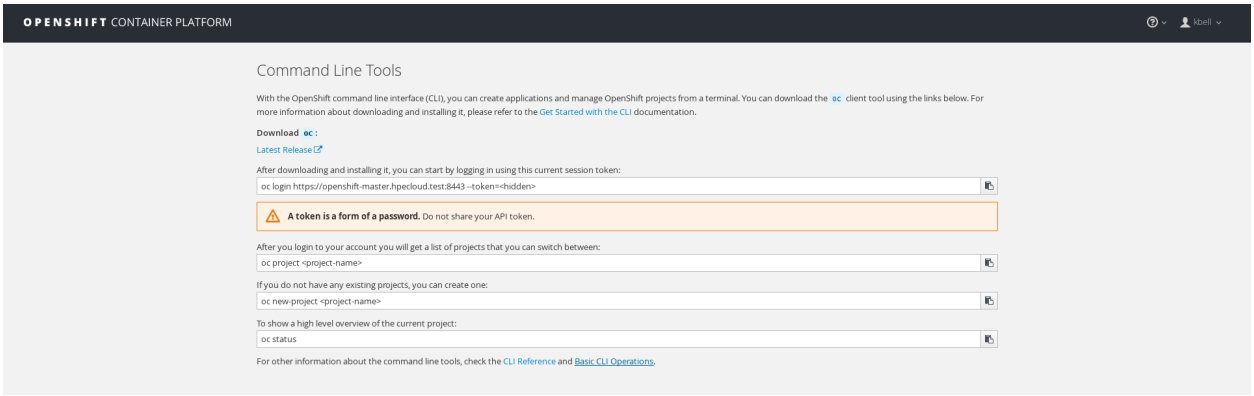


Figure 33: Command Line Tools

Using the *oc login* command, log into OpenShift using the public URL and the token available from the console command-line interface described above.

```
oc login https://openshift-master.hpecloud.test:8443 --
token=zQN6GDIPOWpklX-Lq4DCCL2CMRRI5jdCYEYIjW5GofA
```

CakePHP Sample Application

Create a new Project

Use the *oc new-project* command to create a new OpenShift Container Platform project for the application.

```
$ oc new-project my-app
Now using project "my-app" on server "https://openshift-
master.hpecloud.test:8443".
You can add applications to this project with the 'new-app' command. For
example, try:
    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-
ex.git
to build a new example application in Ruby.
```

Create a new Application

Create a new application by executing the *oc new-app* command and specifying the git repository for the cakephp application.

```
$ oc new-app https://github.com/openshift/cakephp-ex.git --name=cakephp
```

```
--> Found image 79f7d44 (3 weeks old) in image stream "openshift/php"
under tag "7.0" for "php"
    Apache 2.4 with PHP 7.0
    -----------------------
    Platform for building and running PHP 7.0 applications
    Tags: builder, php, php70, rh-php70
    * The source repository appears to match: php
    * A source build using source code from
https://github.com/openshift/cakephp-ex.git will be created
      * The resulting image will be pushed to image stream
"cakephp:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "cakephp"
    * Port 8080/tcp will be load balanced by service "cakephp"
      * Other containers can access this service through the hostname
"cakephp"
--> Creating resources ...
    imagestream "cakephp" created
    buildconfig "cakephp" created
    deploymentconfig "cakephp" created
    service "cakephp" created
--> Success
    Build scheduled, use 'oc logs -f bc/cakephp' to track its progress.
    Run 'oc status' to view your app.
```

Expose the Application

Create a route to expose the application using the subdomain that was created during the OpenShift Container Platform deployment and the *oc expose* command. In our example the subdomain specified was *paas.hpecloud.test* .

```
$ oc expose svc cakephp
```

Access the Application

The cakephp application can now be accessed via a browser, in our example the URL for the cakephp application is http://cakephp-my-app.paas.hpecloud.test/.

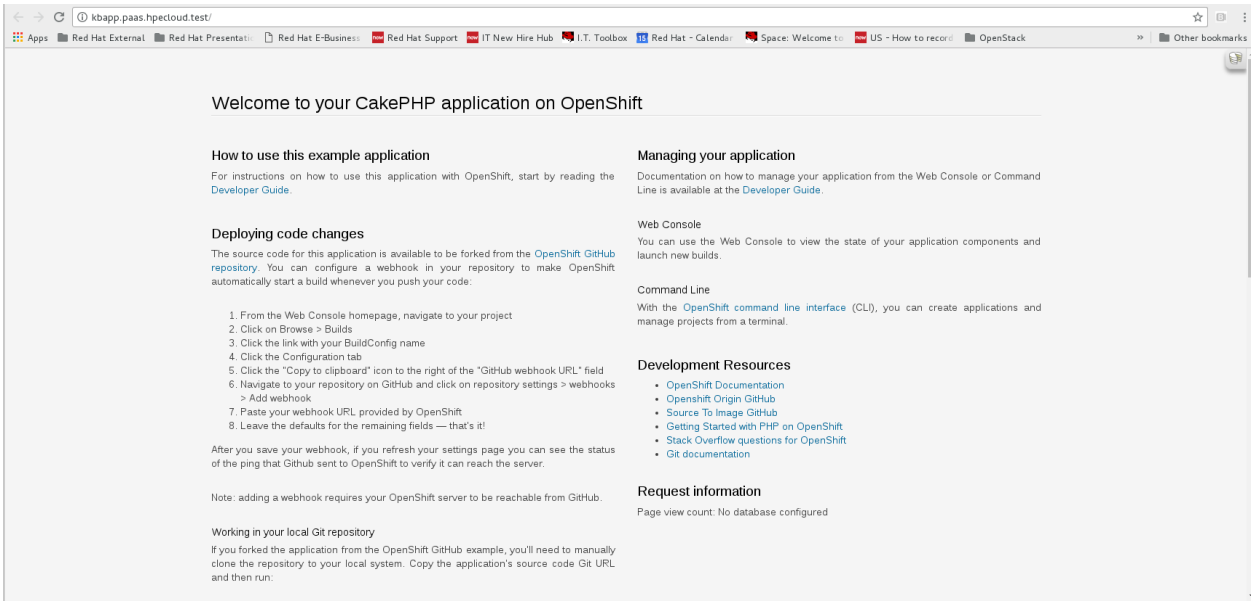


Figure 34: Cake Sample Application

CHAPTER 10. CONCLUSION

This reference architecture provides installation and configuration details for deploying Red Hat OpenShift Container Platform 3.5 on HPE ProLiant DL rack mounted servers. The Red Hat OpenShift Container platform deployment is automated from end to end using Ansible Tower playbooks and workflows, providing a single click deployment from baremetal to a fully configured Red Hat OpenShift Container Platform cluster consisting of three OpenShift Master nodes, three OpenShift infrastructure and application nodes, and three Gluster Container-native storage nodes. The Ansible playbooks used in this reference architecture interact with HPE OneView 3.1 to create server profiles for the baremetal servers, configure the local storage, and update the server firmware to the latest Service Pack for ProLiant (SPP). Ansible playbooks are used to provision the servers with Red Hat Satellite. Red Hat Enterprise Linux 7.3 is installed on each of the nodes and access content repositories for Red Hat OpenShift Container Platform 3.5 and Red Hat Gluster Storage from Red Hat Satellite. Additionally, Red Hat Satellite provide DNS and DHCP services to the deployment. The reference architecture also describes post deployment activities that verify that the Red Hat OpenShift Container Platform is operating correctly. These post deployment validation and configuration processes include assessing the health of the OpenShift and Gluster services, and deploying a sample application (cakephp).

APPENDIX A. BILL OF MATERIALS

Name	Description	Part Number	Quantity
Master Nodes	**DL360p Gen9 8SFF**	**AIOP-13024**	**3**
ProLiant DL360 Gen9	HPE ProLiant DL360 Gen9 8SFF Configure-to-order Server HPE Dynamic Smart Array B140i 8-Bay SFF Drive Cage	755258-B21	1
Processor 1	Intel Xeon E5-2699v3 (2.6GHz/18-core/45MB/145W) FIO Processor Kit	780003-L21	1
Processor 2	Intel Xeon E5-2699v3 (2.6GHz/18-core/45MB/145W) FIO Processor Kit	780003-B21	1
Memory for 1st processor	16GB (1x16GB) Dual Rank x4 DDR4-2133 CAS-15-15-15 Registered Memory Kit	726719-B21	2
Memory for 2nd processor	16GB (1x16GB) Dual Rank x4 DDR4-2133 CAS-15-15-15 Registered Memory Kit	726719-B21	2
FlexibleLOM	HPE Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter	665243-B21	1
ALOM	2 port 10Gb NIC	727055-B21	1

Name	Description	Part Number	Quantity
Power Supply	HPE 800W Flex Slot Platinum Hot Plug Power Supply Kit	720479-B21	2
Hard Drives	HPE 1.2TB 6G SAS 10K rpm SFF (2.5-inch) SC Dual Port Enterprise 3yr Warranty Hard Drive	718162-B21	2
Storage Controller	HPE Smart Array P440ar/2GB FBWC 12Gb 2-ports Int FIO SAS Controller	749974-B21	1
Rail Kit	HPE 1U SFF Ball Bearing Rail Kit	663201-B21	1
SATA Cable	HPE DL360 SFF Embed SATA Cable	766207-B21	1
Application and Infrastructure Nodes	**DL360p Gen9 8SFF**	**AIOP-13024**	**3**
ProLiant DL360 Gen9	HPE ProLiant DL360 Gen9 8SFF Configure-to-order Server HPE Dynamic Smart Array B140i 8-Bay SFF Drive Cage	755258-B21	1
Processor 1	Intel Xeon E5-2690v3 (2.6GHz/12-core/30MB/135W) Processor	755396-B21	1

Name	Description	Part Number	Quantity
Processor 2	Intel Xeon E5-2690v3 (2.6GHz/12-core/30MB/135W) FIO	755396-L21	1
Memory for 1st processor	16GB (1x16GB) Dual Rank x4 DDR4-2133 CAS-15-15-15 Registered Memory Kit	726719-B21	2
Memory for 2nd processor	16GB (1x16GB) Dual Rank x4 DDR4-2133 CAS-15-15-15 Registered Memory Kit	726719-B21	2
FlexibleLOM	HPE Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter	665243-B21	1
ALOM	2 port 10Gb NIC	727055-B21	1
Power Supply	HPE 800W Flex Slot Platinum Hot Plug Power Supply Kit	720479-B21	2
Hard Drives	HPE 1.2TB 6G SAS 10K rpm SFF (2.5-inch) SC Dual Port Enterprise 3yr Warranty Hard Drive	718162-B21	2
Storage Controller	HPE Smart Array P440ar/2GB FBWC 12Gb 2-ports Int FIO SAS Controller	749974-B21	1
Rail Kit	HPE 1U SFF Ball Bearing Rail Kit	663201-B21	1

Name	Description	Part Number	Quantity
SATA Cable	HPE DL360 SFF Embed SATA Cable	766207-B21	1
Container-native storage Nodes	**DL380 Gen9 24SFF**		**3**
ProLiant DL380 Gen9	HP ProLiant DL380 Gen9 24SFF Configure-to-order Server HP Dynamic Smart Array B140i 24-Bay SFF Drive Cage	767032-B21	1
Storage	DL380 Gen9 2SFF Front/Rear SAS/SATA Kit	724864-B21	1
Processor	Intel Xeon E5-2643v3 (3.4GHz/6-core/20MB/135W) FIO Processor Kit	719057-L21	1
Memory	8GB (1x8GB) Single Rank x4 DDR4-2133 CAS-15-15-15 Registered Memory Kit	726718-B21	8
FlexibleLOM	HP Ethernet 10Gb 2-port 560FLR-SFP+ FIO Adapter	665243-B21	1
ALOM	2 port 10Gb NIC	727055-B21	1

Name	Description	Part Number	Quantity
Power Supply	HP 800W Flex Slot Platinum Hot Plug Power Supply Kit	720479-B21	2
Hard Drives	HP 1.2TB 6G SAS 10K rpm SFF (2.5-inch) SC Dual Port Enterprise 3yr Warranty Hard Drive	718162-B21	12
SSD Hard Drives	HP 400GB 12G SAS Write Intensive SFF 2.5-in SC 3yr Wty Solid State Drive	802582-B21	1
Storage Controller	HP Smart Array P440ar/2GB FBWC 12Gb 2-ports Int FIO SAS Controller	749974-B21	1
SAS Expander	HP 12G DL380 Gen9 SAS Expander Card	727250-B21	1
Rail Kit	HP 2U SFF Ball Bearing Rail Kit	720863-B21	1
Rack Infrastructure and Networking			
PDU	NA/JPN - HP 4.9kVA 208V 20out NA/JP bPDU	H5M58A	4
OpenStack Mgmt Switch	HP FF 5930 32QSFP+ Switch	JG726A	2

Name	Description	Part Number	Quantity
IPMI/iLO Switch	HP FF 5700-48G-4XG-2QSFP+ Switch	JG894A	1
Switch power supplies	HP A58x0AF 650W AC Power Supply	JC680A	2
Switch power supplies	HP A58x0AF 300W AC Power Supply	JG900A	2
Switch fan trays	HP X712 Bck(pwr)-Frt(prt) HV Fan Tray	JG553A	2
Switch fan trays	HP A58 X0Af Bck(pwr)-Frt(prt) Fan Tray	JC682A	2
Server split-out cables	HP X240 QSFP+ 4x10G SFP+ 5m DAC Cable	JG331A	9
40Gb DAC cable for IRF	HP X240 40G QSFP+ QSFP+ 5m DAC Cable	JG328A	1

Table 10: Bill of Materials

APPENDIX B. REVISION HISTORY

Revision 1.0-0	**2017-07-25**	**DC**
Revision 1.02-0	**2017-08-09**	**kbell**

- **Changed all instances of Container Native Storage to Container-native storage.**
- **Updated Figure 1.**

Revision 1.01-0	**2017-07-28**	**kbell**

- **Added note indicating scripts and playbooks referenced in this document are not supported by Red Hat**

Revision 1.0-0	**2017-07-25**	**kbell**

- **Initial creation of document**