Reference Architectures 2019

Deploying Red Hat OpenShift Container Platform 3.11 on Red Hat OpenStack Platform 13

Last Updated: 2019-07-22

Reference Architectures 2019 Deploying Red Hat OpenShift Container Platform 3.11 on Red Hat OpenStack Platform 13

Jacob Liberman
jliberma@redhat.com

Legal Notice

Abstract

The purpose of this document is to provide guidelines and considerations for deploying Red Hat OpenShift Container Platform on Red Hat OpenStack Platform 13.

Table of Contents

EXECUTIVE SUMMARY

Organizations across the globe are looking to rapidly develop innovative software applications in Hybrid and Multi-Cloud in order to achieve competitive advantage and ensure customer satisfaction. Many of these software applications have to be deployed in an on-premises private cloud or in a co-location facility for various reasons (for example, security and compliance, data affinity, performance, among others). The IT organizations responsible for operating the private cloud desire it to be simple, agile, flexible, secure, cost efficient, and part of their overall Hybrid and Multi-Cloud architecture.

This Reference Architecture showcases a prescriptive and pre-validated private cloud solution from Red Hat that allows you to run IT as a Service (ITaaS), and provides rapid provisioning and lifecycle management of containerized apps, virtual machines (VMs), and associated application and infrastructure services for software developers, data scientists, solution architects, among others. The key architecture components of this solution include Red Hat OpenShift Container Platform, Red Hat OpenStack Platform, and Red Hat Ceph Storage.

Extensive testing was performed at the Red Hat labs to validate the design, deployment, and management best practices, so that organizations can quickly deploy the solution and achieve faster time to value. This pre-validation by Red Hat can help mitigate risk in deployments, and can eliminate most of the time required to develop the Day 1-2 operations best practices.

CHAPTER 1. SOLUTION OVERVIEW

This section provides an overview of the target use cases, benefits to IT and business, and a high level overview of the end-to-end solution architecture.

1.1. TARGET USE CASES

This Reference Architecture is valuable for Enterprise, Telco IT, Government, and IT Service Provider organizations that intend to deploy a Private Cloud solution with programmable Infrastructure as a Service (IaaS), Containers as a Service (CaaS), and Platform as a Service (PaaS) capabilities. Some of the key targeted use cases with this solution are software developer cloud, and web, mobile, AI/ML, and analytics workloads.

1.2. SOLUTION BENEFITS FOR IT AND BUSINESS

Listed below are the key benefits of this solution for IT organizations, and how it derives value for the business:

- Faster time to value and innovation as this Reference Architecture will save a lot of time that may have historically been required to design, deploy, and test the full solution stack comprised of several Red Hat products (such as OpenShift, OpenStack, RHEL, and Ceph) and products from our ecosystem partners.
- Simplify and de-risks deployments as all the desired best practices for the full solution stack are already pre-validated by Red Hat, resulting in a highly prescriptive solution to ensure success.
- Cost efficiency as the Reference Architecture is based on open source technologies, fully supported by Red Hat.

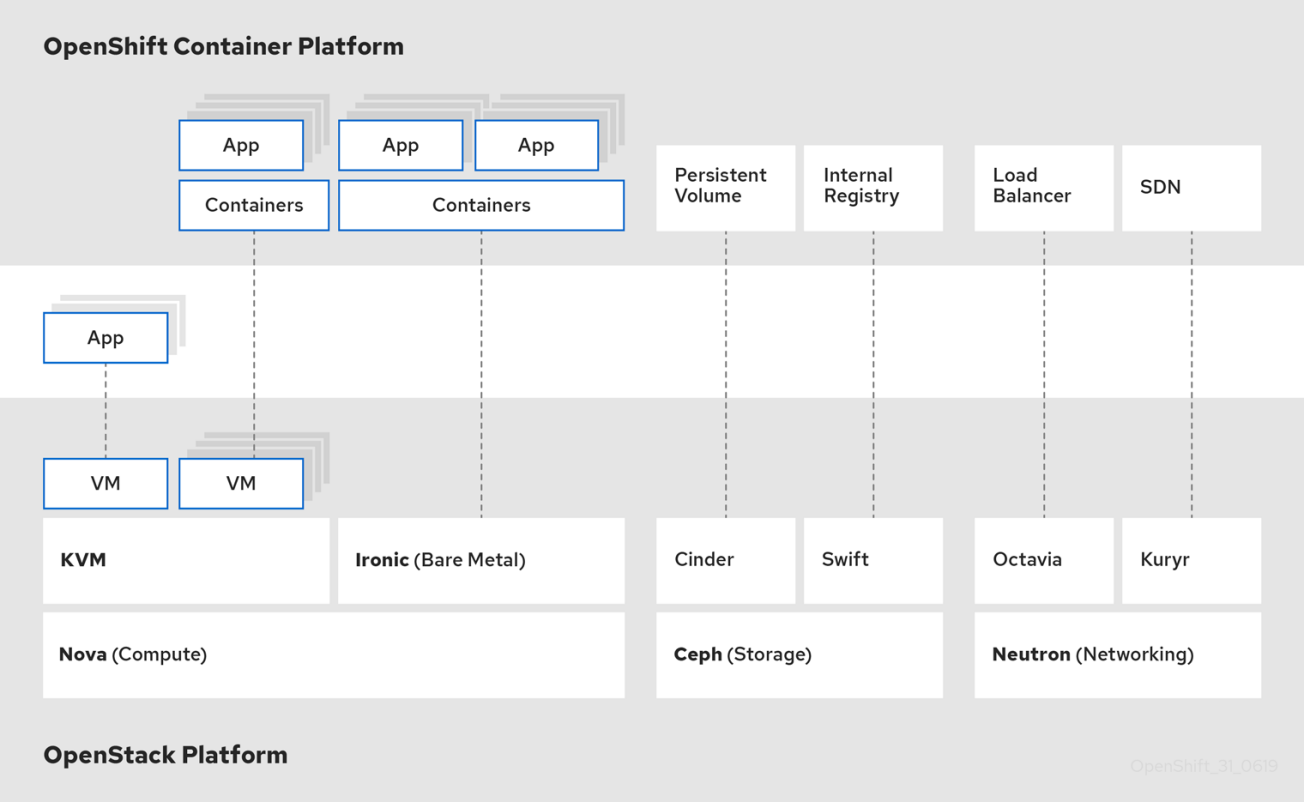
CHAPTER 2. ARCHITECTURE OVERVIEW

This is a reference architecture for running OpenShift Container Platform 3.11 on Red Hat OpenStack Platform 13. In this reference architecture, Red Hat OpenStack Platform is deployed to physical servers; OpenShift Container Platform is deployed to virtual machines running on Red Hat OpenStack Platform's hypervisor. The architecture is highly available and suitable for production.

Both OpenShift Container Platform and OpenStack Platform are modular and configurable, and there are many supported ways to install OpenShift on OpenStack. Red Hat field consultants developed the recommendations in this reference architecture based on their experience deploying to production. This document describes the reference architecture as it was deployed and tested in a lab environment. It also shares important design considerations and key integrations between the products.

Step by step instructions for installing Red Hat OpenStack Platform and Red Hat OpenShift Container Platform is beyond the scope of this document. For detailed installation instructions, see the OpenStack Platform 13 Director Installation and Usage Guide and the OpenShift Container Platform 3.11 product documentation.

Figure 1:



2.1. RELATIONSHIP BETWEEN OPENSHIFT AND OPENSTACK

Figure 1 shows a block diagram of OpenShift Container Platform running on OpenStack Platform. Starting from the bottom, Red Hat OpenStack Platform is deployed on commodity hardware; OpenShift Container Platform is installed on either virtual machines or bare metal servers provisioned by OpenStack. Containerized applications run in OpenShift alongside applications running on OpenStack virtual machines.

The relationship between OpenStack and OpenShift is complementary: OpenStack exposes resources that OpenShift consumes. It provides OpenShift with compute, storage, and networking infrastructure plus additional resources such as self-service load balancers and encryption. OpenShift runs its

containerized applications on the infrastructure provisioned by OpenStack. The products are tightly integrated, allowing OpenShift to consume OpenStack resources on demand and without user intervention.

Figure 2:

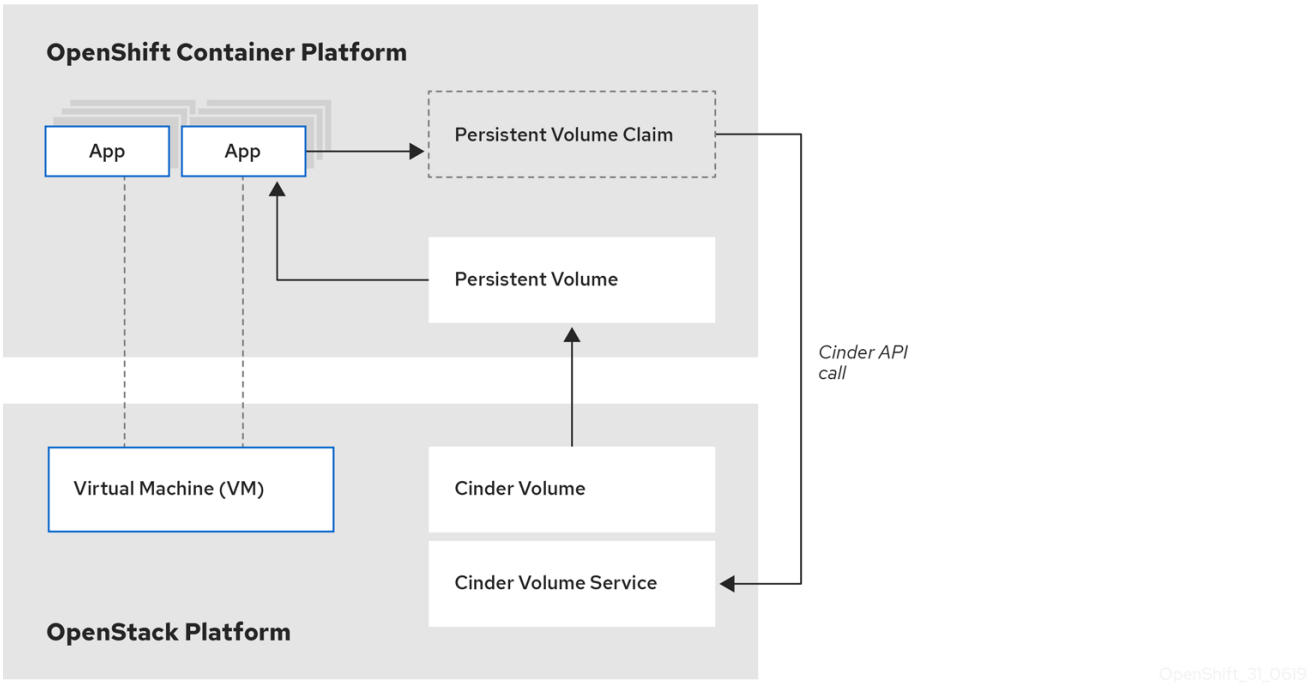


Figure 2 illustrates the relationship between OpenShift and OpenStack by showing a typical interaction between a containerized application and the infrastructure; this example relates to storage. When an application running on OpenShift needs a persistent volume, it submits a persistent volume claim to the OpenShift API. This OpenShift API call is translated into a Cinder API call to create a volume. When the volume is ready it is presented back to OpenShift, and attached to the requesting pod. The persistent volume claim only needs to include the volume size and access mode. The backend implementation details of how and where the volume is created are handled by OpenStack. The OpenShift API abstracts them from the user making the resource claim. This pattern of interaction is repeated for other OpenShift infrastructure requests to OpenStack.

CHAPTER 3. DESIGN CONSIDERATIONS

Consider the following questions when adapting this reference architecture to your environment:

- What installation tooling will you use?
- Is high availability required?
- Which storage and network backends will you use?
- How will DNS be provided?
- What security and authentication features are needed?

This section describes how this reference architecture addresses each of these design considerations. The recommendations in this reference architecture were developed by Red Hat field consultants for deploying OpenShift on OpenStack in production.

3.1. INSTALLATION TOOLS

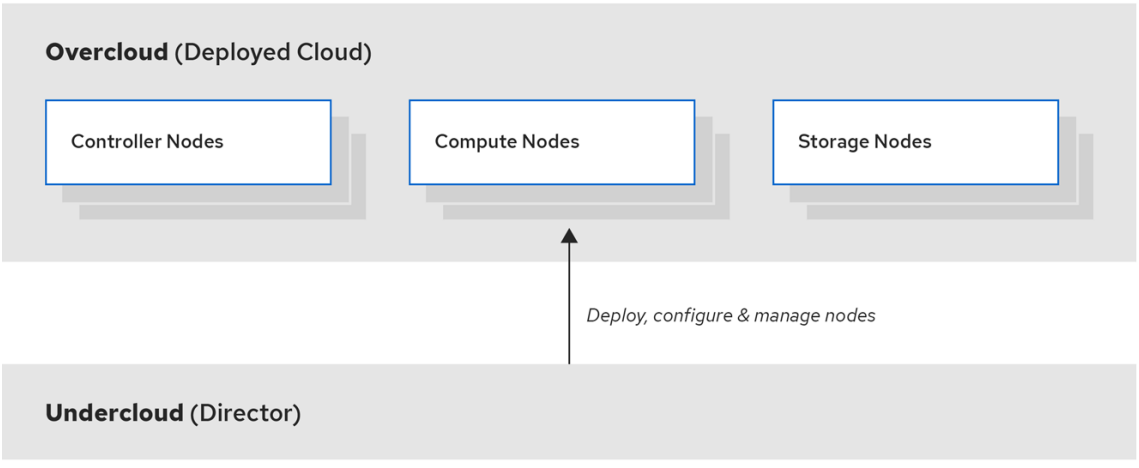
Perform the installation using the tools recommended and supported by Red Hat: Red Hat OpenStack Platform director and openshift-ansible.

3.1.1. Red Hat OpenStack Platform director

Red Hat OpenStack Platform director is a management and installation tool based on the OpenStack TripleO (OpenStack on OpenStack) project. This reference architecture uses director to install Red Hat OpenStack Platform.

The fundamental concept behind Red Hat OpenStack Platform director is that there are two clouds. The first cloud (called the *undercloud*) is a standalone OpenStack deployment. The undercloud can be deployed on a single physical server or virtual machine. The administrator uses the undercloud's OpenStack services to define and deploy the production OpenStack cloud. Director is also used for day two management operations, such as applying software updates and upgrading between OpenStack versions.

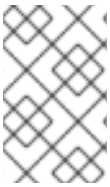
Figure 3:



The second cloud (called the *overcloud*) is the full-featured production environment deployed by the undercloud. The overcloud is comprised of physical servers with various roles:

- *Controller nodes* - run the OpenStack API endpoints. They also store OpenStack's stateful configuration database and messaging queues.
- *Compute nodes* - run virtual machine hypervisors. They host the computing resources allocated for user workloads.
- *Storage nodes* - provide block, object, or software defined storage for the user workloads.

Figure 3 depicts the relationship between the undercloud, overcloud, and OpenStack nodes. Red Hat OpenShift Container Platform runs on the overcloud.



NOTE

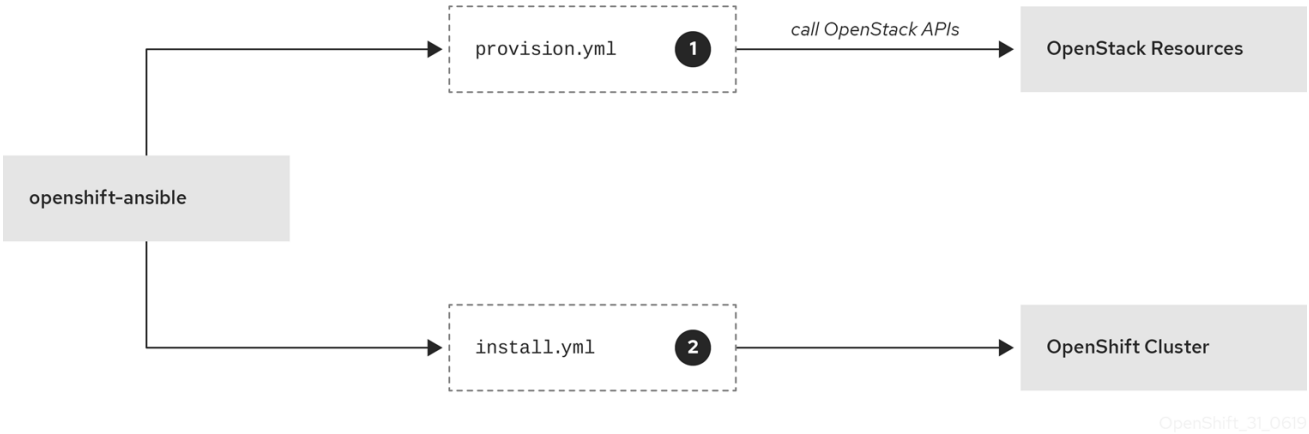
Director is required for all production Red Hat OpenStack Platform deployments. Red Hat engineering's tested and validated configuration settings are embedded into director's deployment tooling.

3.1.2. Openshift-ansible

Once OpenStack is installed, virtual machines are built within OpenStack to run OpenShift. The virtual machines are configured using Ansible roles from openshift-ansible. *Openshift-ansible* is a set of Ansible playbooks that orchestrate complex deployment tasks including:

- Configuring the container runtime environment on virtual machines.
- Provisioning storage for an internal registry.
- Configuring the OpenShift SDN.
- Connecting to authentication systems.

Figure 4:



Openshift-ansible deploys OpenShift on OpenStack using two playbooks:

- *provision.yml* - deploys the OpenStack virtual machines. It uses the Ansible cloud provider module to build OpenStack resources using calls directly to the OpenStack Heat API. The *all.yml* Ansible vars file defines how OpenStack should be configured.
- *install.yml* - installs the OpenShift cluster on the virtual machines. It uses the *OSEv3.yml* Ansible vars to define how OpenShift should be deployed.

More information on using openshift-ansible can be found in the official OpenShift Container Platform 3.11 documentation.

Openshift-ansible uses a dynamic Ansible inventory script to issue commands against roles. The inventory is automatically generated as output from **provision.yml**. The following command output shows an OpenShift host list generated from the dynamic inventory:

```
(shiftstack) [cloud-user@bastion openstack]$ ./inventory.py --list | jq .OSEv3.hosts

[

"master-1.openshift.example.io",

"app-node-1.openshift.example.io",

"app-node-2.openshift.example.io",

"infra-node-1.openshift.example.io",

"master-2.openshift.example.io",

"master-0.openshift.example.io",

"app-node-0.openshift.example.io",

"infra-node-0.openshift.example.io",

"infra-node-2.openshift.example.io"

]
```

Openshift-ansible can also deploy OpenShift directly onto physical servers using the OpenStack Ironic API. This reference architecture deploys to virtual machines as that is the more common deployment model.

3.2. HIGH AVAILABILITY

High availability is a requirement for any production deployment. A crucial consideration for high availability is the removal of single points of failure. This reference architecture is highly available at both the OpenStack and OpenShift layers.

3.2.1. OpenStack HA

OpenStack Platform Director deploys three controller nodes. Multiple instances of the OpenStack services and APIs run simultaneously on all three controllers. HAproxy load balances connections across the controller API endpoints to ensure service availability. The controllers also run the OpenStack state database and message bus. A Galera cluster protects the state database. RabbitMQ queues are duplicated across all nodes to protect the message bus. This is the default level of high availability enforced by Red Hat OpenStack Platform director.

3.2.2. OpenShift HA

OpenShift is also deployed for high availability. In this reference architecture the *etcd state database* is co-located across the master nodes. **etcd** requires a minimum of three nodes for high availability.

This reference architecture also uses three infrastructure nodes. Infrastructure nodes host OpenShift infrastructure components such as the registry, containers for log aggregation, and metrics. A minimum of three infrastructure nodes are needed for high availability when using a sharded aggregated logging database, and to ensure service interruptions do not occur during a reboot.

In deployments with three or more OpenStack Compute nodes, OpenShift should be configured to use node anti-affinity. This feature helps ensure high availability by preventing virtual machines of the same role from running on the same Compute node. Anti-affinity prevents a single Compute node failure from bringing down the OpenShift cluster.

Figure 5:

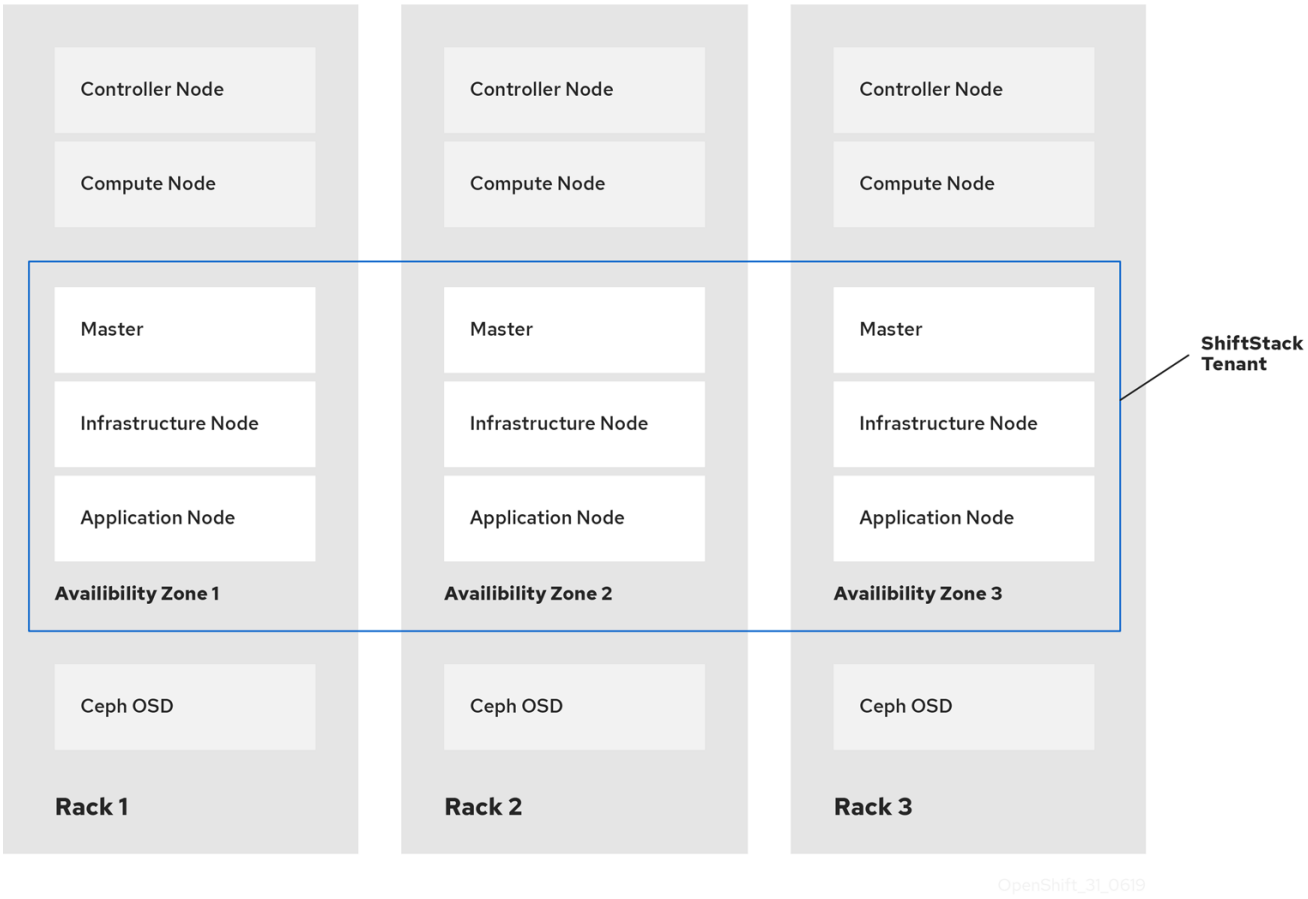


Figure 5 shows an example of a highly available OpenShift on OpenStack deployment. The OpenStack compute nodes and Ceph OSDs are grouped into availability zones on a per-rack basis. The virtual machines are all members of the same OpenStack tenant. Affinity rules spread the virtual machines across the physical compute nodes by role.

3.2.3. Hardware HA

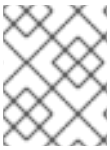
You should also take care to eliminate single points of failure at the hardware layer. Hardware fault tolerance recommendations implemented in this architecture include:

- RAID on server internal hard disks.
- Ceph OSD disks should be configured as RAID 0 or without RAID. The OSDs are protected by Ceph's native data replication.
- Redundant server power supplies connected to different power sources.

- Bonded network interfaces connected to redundant network switches. The following code sample shows a network bond definition for an OpenStack controller:

```
- type: ovs_bridge

name: br-vlan

members:

- type: linux_bond

name: bond2

bonding_options:

get_param: BondInterfaceOvsOptions

members:

- type: interface

name: nic4

primary: true

- type: interface

name: nic5

...

BondInterfaceOvsOptions: 'mode=1 miimon=150'
```

If the OpenStack deployment spans multiple racks, the racks should be connected to different PDUs. OpenStack Compute nodes should be divided into availability zones by PDU.



NOTE

A complete description of how to configure hardware fault tolerance is beyond the scope of this document.

3.3. STORAGE AND NETWORKING

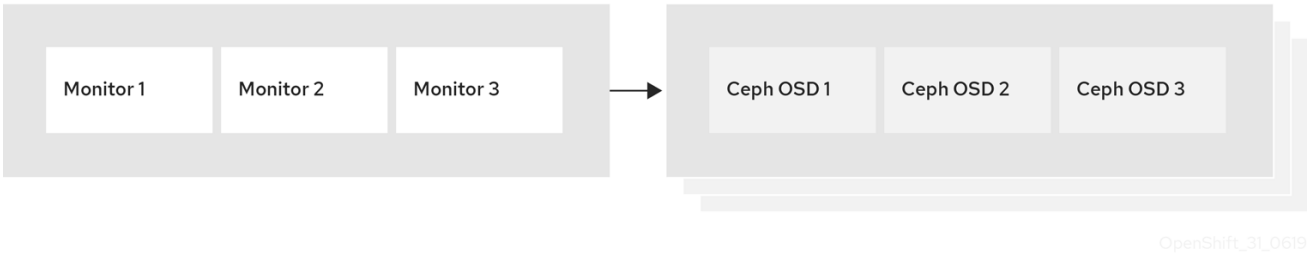
Plan the storage and networking carefully. Select storage and networking backends that meet the OpenShift application's needs while minimizing complexity. Both Red Hat OpenStack Platform and OpenShift Container Platform have independent and mature storage and networking solutions. However, combining the native solutions for each platform can increase complexity and unwanted performance overhead. Avoid duplicate storage or networking components. For example, do not run an OpenShift SDN on top of an OpenStack SDN, and do not use OpenShift Cluster Storage over Ceph.

3.3.1. Red Hat Ceph Storage

Red Hat Ceph Storage provides scalable cloud storage for this reference architecture. Ceph is tightly integrated with the complete Red Hat cloud software stack. It provides block, object, and file storage capabilities.

Ceph cluster servers are divided into Monitors and Object Storage Device (OSD) nodes. Ceph monitors run the monitor daemon, which keeps a master copy of the cluster topology. Clients query the Ceph monitors in order to read and write data to the cluster. Ceph OSD nodes store data. The data are replicated across the physical disks within the nodes.

Figure 6:



A minimum of three Ceph monitors and three or more Ceph OSD nodes are needed to ensure high availability in production. This is depicted in figure 6. Other than recommending the minimum number of Ceph nodes for high availability, Ceph hardware sizing guidelines are beyond the scope of this document. They vary depending on the workload size and characteristics. Refer to the Ceph Hardware Selection Guide for additional information.

This reference architecture also provides S3-compatible object storage using the Ceph Rados Gateway. A minimum of three Object Gateway nodes are required for external access to Ceph object storage.

NOTE

It is recommended that all Ceph nodes run on dedicated physical servers.

3.3.2. Neutron networking

The OpenStack Neutron API provides a common abstraction layer for various backend network plugins. Red Hat OpenStack Platform 13 supports multiple backend network plugins, including ML2 OVS, OVN, and commercial SDNs. This reference architecture uses the default ML2 OVS plugin. VXLAN encapsulation carries layer 2 traffic between nodes. All L3 traffic is routed through centralized Neutron agents running on the controller nodes.

A range of externally accessible floating IP addresses are bridged to the tenant instances using a provider network. The floating IP addresses are used by remote clients to access exposed OpenShift services, the OpenShift API endpoint, and the web console.

3.3.3. OpenShift SDN

The OpenShift SDN connects pods across all node hosts. It provides a unified cluster network. OpenShift natively provides several options for inter-pod communication. These include OVS multi-tenant and network policy. More information on OpenShift 3 SDN concepts can be found in the OpenShift 3 SDN documentation.

This reference architecture uses Kuryr for OpenShift networking. Kuryr allows OpenShift pods direct access to the underlying OpenStack Neutron networks. It directly plumbs OpenShift pods and OpenStack virtual machines to the same software defined networks. Kuryr is described further in the integration section of this document.

3.4. DNS

OpenShift Container Platform requires a fully functional DNS server. This section describes design considerations related to providing OpenShift with DNS through OpenStack.

3.4.1. OpenShift DNS

Internally, OpenShift uses DNS to resolve container names. OpenShift nodes run local *dnsmasq* services to resolve the internal addresses.

OpenShift also requires external DNS; openshift-ansible resolves OpenShift node hostnames while validating certificate signing requests during installation. The OpenShift console and containerized applications exposed through the external router must also be externally resolvable through DNS.

Openshift-ansible uses floating IP addresses for the externally resolvable OpenShift addresses, this is not ideal for statically configured external DNS. Any delete or redeploy of the OpenShift environment requires a change to the external DNS records. Therefore, in this reference architecture openshift-ansible is configured to dynamically push the OpenShift console and application wildcard addresses to an external server using **nsupdate**.

3.4.2. OpenStack DNS

OpenStack can provide DNS for OpenShift in multiple different ways. By default, OpenStack is configured for a provider-type environment. Tenants are expected to bring their own DNS. The external DNS server may exist outside of OpenStack, or it can be deployed within the tenant, forwarding addresses it cannot resolve to an external DNS server. The external DNS server address is pushed to the OpenStack instances using the Neutron subnet configuration for the tenant.

Red Hat OpenStack Platform 13 also supports native DNS resolution using Neutron; this is not enabled by default. When enabled, Neutron associates DNS names with ports. The port names are internally resolvable by **dnsmasq** running on the OpenStack hypervisors. Dnsmasq can be configured to forward addresses it cannot resolve to a remote DNS server or to whatever external DNS server is used by the OpenStack hypervisor. Note that when this approach is used, the default DNS suffix is changed for all OpenStack tenants, which does not cause a problem. Namespace isolation allows the same name to be used in multiple tenants without any issues.

This reference architecture implements the first approach. A BIND server was deployed within the tenant to provide local DNS resolution. “Bring your own DNS” is the more common case. The internal Neutron DNS approach involves changing the default OpenStack configuration deployed by Director. If internal Neutron DNS resolution is not configured during the initial OpenStack deployment, enabling it later can be an invasive process.

3.5. SECURITY AND AUTHENTICATION

This section shares design considerations related to security and authentication when running OpenShift on OpenStack.

OpenShift and OpenStack are enterprise open source software, and both support Role Based Access Control (RBAC) and flexible options for integrating with existing user authentication systems. In addition, both inherit the security Red Hat Enterprise Linux native security features, such as SELinux.

3.5.1. Authentication

By default the OpenStack Identity service (keystone) stores user credentials in its state database, or it can use an LDAP-compliant directory server. Similarly, authentication and authorization in OpenStack can be delegated to another service.

OpenShift master nodes issue tokens to authenticate user requests with the API. OpenShift supports various identity providers including HTPassword and LDAP.

There is no formal integration between OpenShift and OpenStack identity providers and authentication. However, both services can be configured to use the same LDAP directory servers, providing a consistent authentication user experience across the platforms.

3.5.2. Security

The Red Hat OpenStack Platform 13 Security and Hardening Guide recommends practices to help security harden OpenStack. Many of the security practices are embedded into a default Red Hat OpenStack Platform deployment. In the field, Red Hat OpenStack Platform has been security hardened to various levels of standard security compliance, such as ANSI and FedRamp. This reference architecture is not a comprehensive resource for securing OpenStack.

Multiple OpenStack Platform security features are used to help security harden OpenShift in this reference architecture. For example, openshift-ansible creates security groups for every role that only allow port and protocol level access to the services running on that role. In addition, TLS SSL certificates are used to encrypt the OpenStack public API endpoints. The certificate chain can be imported into OpenShift either during installation or manually to allow openshift-ansible to access the encrypted endpoints.

Note that if OpenStack TLS SSL public endpoint encryption is enabled, the certificates must be imported to any hosts issuing commands against the endpoints. This includes the deployment host where openshift-ansible is run.

3.5.3. OpenStack Barbican

Barbican is the OpenStack Key Manager service API. It is designed for the secure storage, provisioning, and management of secrets such as passwords, encryption keys, and X.509 certificates. Barbican does not appear in this reference architecture, but it was tested during reference architecture development. Potential use cases for Barbican with OpenShift on Openstack include:

- Creating keys for the self-signed Glance images for building OpenShift instances.
- Encrypting object storage buckets for the OpenShift internal registry in a multi-tenant deployment.
- Encrypting Cinder volumes backed by Ceph in a multi-tenant deployment.

Information related to installing and configuring Barbican can be found in the official OpenStack Platform 13 product documentation.

CHAPTER 4. REFERENCE ARCHITECTURE

This chapter of the document describes the deployed reference architecture. It includes architectural details for:

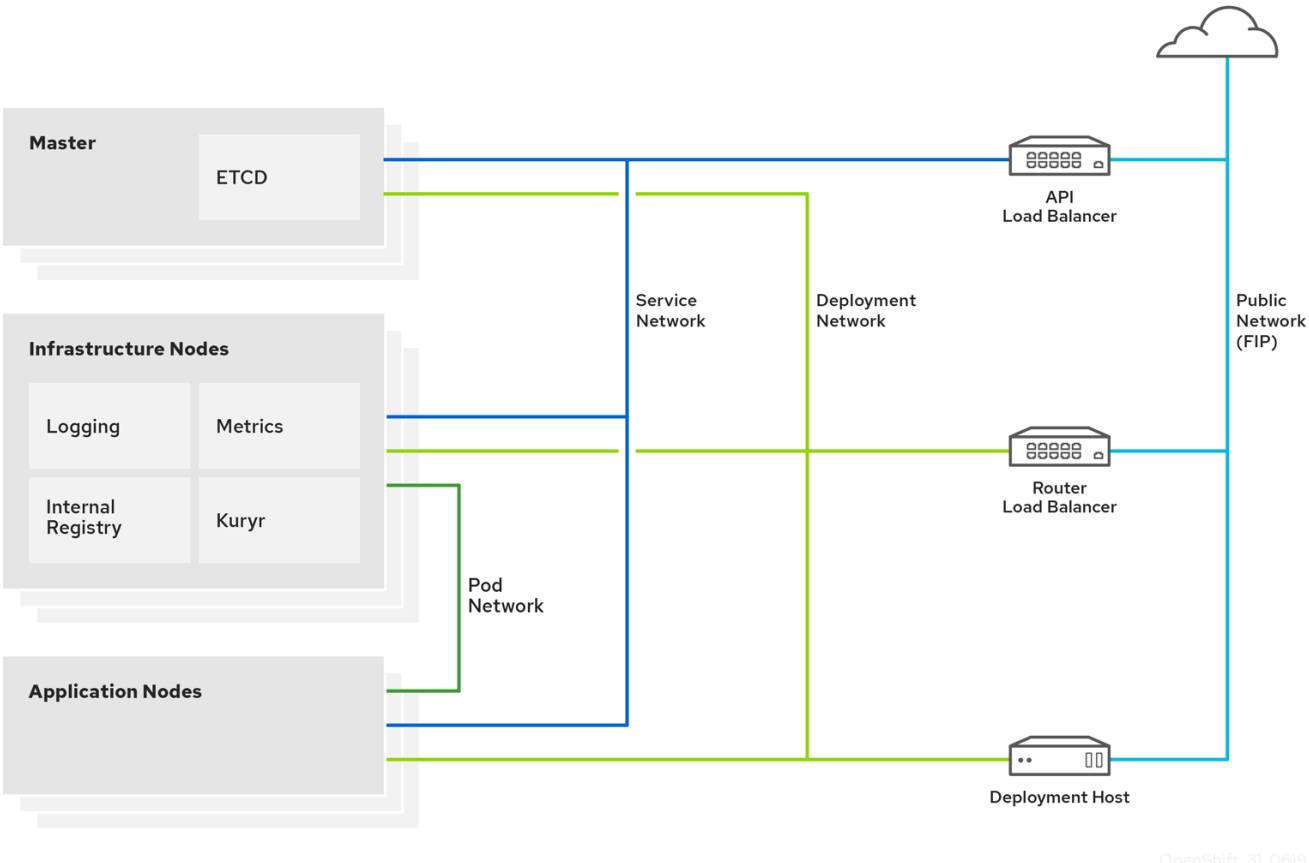
- OpenStack overcloud configuration
- OpenStack tenants (also known as *projects*)
- DNS configuration
- OpenShift roles
- OpenShift service placement
- Storage

The reference architecture does not include step by step instructions for deploying OpenShift or OpenStack, you will need to refer to the official product documentation.

4.1. REFERENCE ARCHITECTURE DIAGRAM

Figure 7 depicts the OpenShift components of the deployed reference architecture for OpenShift on OpenStack. Refer to this diagram to visualize the relationships between OpenShift roles and services, including the underlying OpenStack networks described in the later sections.

Figure 7:



4.2. OPENSTACK TENANT NETWORKING

OpenShift Container Platform is installed into an OpenStack project. Prior to running openshift-ansible, the tenant must meet the prerequisites described in OpenShift Container Platform 3.11: Configuring for OpenStack.

Listed below are the types of tenant networks used in this reference architecture:

- **Public network**: This network is reachable by the outside world. It is an OpenStack provider network that maps to a physical network that exists in the data centre. The public network includes a range of floating IP addresses that can be dynamically assigned to OpenStack instances. It is defined by an OpenStack administrator.

- **Deployment network**: An internal network created by the tenant user. All OpenShift instances are created on this internal network. The deployment network is used to deploy and manage the OpenStack instances that will be running OpenShift. Instances on the deployment network cannot be reached from the outside world unless they are given a floating IP address. **OpenShift pod network**: Created by openshift-ansible during the provisioning phase. It is an internal network for OpenShift inter-pod communication. When Kuryr is enabled, master nodes also have access to the pod network. **OpenShift service network**: The control plane network for OpenShift service communication. It is also created by OpenShift ansible.

A neutron router running in OpenStack routes L3 traffic between the deployment (bastion), pod, and service networks. It also acts as a gateway for instances on internal networks that need to access the outside world.

4.3. DEPLOYMENT HOST

By default, openshift-ansible assigns floating IP addresses to all OpenStack instances during installation. As a result, all instances are accessible from the outside world. You will need to consider whether this approach is acceptable in your deployment.

This reference architecture uses a deployment host. The deployment host has a network interface on the internal network as well as an externally accessible floating IP. The tenant user runs openshift-ansible from the deployment host.

A deployment host is recommended for production deployments for the following reasons:

- **Security hardening** – OpenShift instances should not be accessible to the outside world or users from other tenants except via the exposed console and router endpoints.

- **Resource allocation** – Floating IP addresses are finite. They should be reserved for tenant users to be allocated when needed.

4.4. DNS CONFIGURATION

DNS is provided by a Bind 9 server accessible using the external (public) network. The DNS server is authoritative for the **openshift.example.io** domain. This domain resolves the public address records specified in openshift-ansible and updated using **nsupdate**. An example public **nsupdate** key is shown in the following example:

```
openshift_openstack_external_nsupdate_keys:

public:

key_secret: "<secret_key>"
```

```
key_name: "public-openshift.example.io"

key_algorithm: "HMAC-SHA256"

server: "192.168.122.252"
```

The DNS server is named **ns1.example.io**. It is configured to forward addresses it cannot resolve to an external DNS server.

The openshift instance hostnames resolve to addresses on the **bastion_net** network. A record for the deployment server must be added manually, as openshift-ansible does not add it automatically. The following example shows a zone file populated by openshift-ansible:

```
$ORIGIN apps.openshift.example.io.

* A 192.168.122.155

$ORIGIN openshift.example.io.

app-node-0 A 172.16.1.7

app-node-1 A 172.16.1.6

app-node-2 A 172.16.1.8

console A 192.168.122.166

infra-node-0 A 172.16.1.14

infra-node-1 A 172.16.1.24

infra-node-2 A 172.16.1.19

master-0 A 172.16.1.23

master-1 A 172.16.1.22

master-2 A 172.16.1.15

ns1 A 192.168.122.252

bastion A 172.16.1.26
```

After running **install.yml**, an address record is added for **console.openshift.example.com** that resolves to an externally accessible address on the public network. This address is from the OpenStack floating IP pool.

The **install.yml** playbook also creates a wildcard record named **apps.openshift.example.io**. This address is used to access exposed OpenShift applications from clients outside the internal OpenShift cluster network. The address resolves to a floating IP assigned to the OpenShift router pod.

4.5. OPENSHIFT INFRASTRUCTURE ROLES

OpenShift hosts are either masters or nodes. The masters run the control plane components including the API server, controller manager server, and the etcd state database. Nodes provide the runtime

environment for containers. They run the services required to be managed by the master and to run pods. The master schedules pods to be run on the nodes.

This reference architecture uses the following infrastructure node roles:

Role	Count	Flavor	RAM	vCPU	Disk
master	3	m1.master	16384	4	45
infra-node	3	m1.node	8192	2	20
app-node	3	m1.node	8192	2	20

The RAM, CPU, and disk recommendations in this reference architecture should be considered minimums, so adjust them as needed. Hawkular metrics, the Prometheus cluster monitoring service, and the EFK stack for log aggregation in particular require additional disk and memory based on polling intervals and storage retention length. Specific guidelines for configuring these services and environment specific and beyond the scope of this document.

Red Hat OpenStack Platform can be configured to boot instances from persistent Cinder volumes backed by Ceph RBD. Note that this was not done in this reference architecture. The images for instances booted from persistent volumes backed by Ceph RBD should be converted to RAW format prior to booting the instances. Instances booted from persistent volumes backed by Ceph RBD enjoy additional data protection features such as data live migration, snapshots, and mirroring.

In this reference architecture, three master nodes are deployed for high availability to help ensure the cluster has no single point of failure. An Octavia load balancer balances the loads between API master endpoints. The controller manager server runs in an active-passive configuration with one instance elected as a cluster leader at one time.

Etcd is a distributed key-value store that OpenShift Container Platform uses for configuration. The OpenShift API server onsults the etcd state database for node status, network configuration, secrets, and more. In this reference architecture the etcd state database is co-located with the master nodes in order to optimize the data path between the API server and the database.

This reference architecture also uses infrastructure nodes. These are OpenShift nodes that run the OpenShift Container Platform infrastructure components including routers, the cluster monitoring operator, the registry endpoint, and the kuryr controller. Two infrastructure nodes are required for high availability. However, three nodes are deployed in this reference architecture to support the logging and metrics services. The sharded ElasticSearch database that backs the container log aggregation pods in particular requires three nodes for high availability.

4.5.1. OpenShift pod placement

Chapter 9, *Appendix A: Pod placement by role* describes where the pods are scheduled to roles in this reference architecture. In some cases, multiple replicas of each pod are scheduled across all nodes of that role. In other cases there are single pods.

For example, the **kuryr-controller** pod runs on a single infrastructure node. **Kuryr-cni** (listener) pods run on all nodes regardless of role. They run as a daemonset that links the containers running on a node to the neutron network.

In addition, the service pods listed in the appendix, application nodes also run the application payload pods defined by user's dpod definitions, deployment configs, and stateful sets.

4.6. OPENSTACK STORAGE

This reference architecture uses OpenStack storage to:

- Back the docker filesystem on instances.
- Provide persistent volumes to pods.
- Store the internal registry.

4.6.1. Instance volumes

A Cinder persistent volume is provisioned for each master and node by the openshift-ansible installer. The volume is mounted to **/var/lib/docker**, as the following example demonstrates.

```
[openshift@master-0 ~]$ mount | grep dockerlv

/dev/mapper/docker--vol-dockerlv on /var/lib/docker type xfs
(rw,relatime,seclabel,attr2,inode64,prjquota)
```

The container storage volume in this reference architecture is 15 GB and mounted to **/dev/vdb** on all instances. This is handled automatically by openshift-ansible during instance provisioning. The following example shows the list of Cinder volumes attached to each OpenShift instance.

```
[cloud-user@bastion ~]$ openstack volume list -f table -c Size -c "Attached to"

+------+-----------------------------------------------------------------+

| Size | Attached to |

+------+-----------------------------------------------------------------+

| 15 | Attached to infra-node-2.openshift.example.io on /dev/vdb |

| 15 | Attached to infra-node-0.openshift.example.io on /dev/vdb |

| 15 | Attached to infra-node-1.openshift.example.io on /dev/vdb |

| 15 | Attached to master-1.openshift.example.io on /dev/vdb |

| 15 | Attached to master-0.openshift.example.io on /dev/vdb |

| 15 | Attached to master-2.openshift.example.io on /dev/vdb |

| 15 | Attached to app-node-1.openshift.example.io on /dev/vdb |

| 15 | Attached to app-node-0.openshift.example.io on /dev/vdb |

| 15 | Attached to app-node-2.openshift.example.io on /dev/vdb |

+------+-----------------------------------------------------------------+
```

Adjust the size based on the number of size of containers each node will run. You can change the container volume size with the _`**openshift_openstack_docker_volume_size** Ansible variable in **provision.yml**.

4.6.2. Persistent volumes for pods

The openshift-ansible installer automates the creation of a Cinder storage class for dynamic persistent volume creation. The following example shows the storage class:

```
[openshift@master-0 ~]$ oc get storageclass

NAME PROVISIONER AGE

standard (default) kubernetes.io/cinder 1d
```

In this reference architecture the storage class allocates persistent volumes for logging and metrics data storage. Persistent volume claims are issued to the storage class and fulfilled during the installation phase of the openshift-ansible installer. For example:

```
[openshift@master-0 ~]$ oc get pv

CAPACITY ACCESS MODES CLAIM STORAGECLASS

25Gi RWO openshift-infra/metrics-cassandra-1 standard

30Gi RWO openshift-logging/logging-es-0 standard

30Gi RWO openshift-logging/logging-es-1 standard

30Gi RWO openshift-logging/logging-es-2 standard0 ~]$
```

The Cinder storage class access mode is RWO only; this is sufficient for most use cases. RWX (shared) access mode will be addressed in a future reference architecture. There are multiple ways to address the shared use case with OpenShift Container Platform 3.11 and Red Hat OpenStack Platform 13 but they are beyond the scope of this document.

See the OpenStack Container Platform 3.11 Persistent Storage documentation for more information about storage classes and access modes.

4.6.3. Internal registry

OpenShift Container Platform can build container images from source code, deploy them, and manage their lifecycle. To enable this, OpenShift Container Platform provides an internal, integrated container image registry to locally manage images.

The openshift-ansible installer can configure a persistent volume for internal registry storage or use an S3-compatible object store. OpenStack can provide the persistent volume using Cinder, or object storage through Swift backed by Ceph Rados Gateway. The following code block shows the openshift-registry container automatically created by openshift-ansible when **install.yml** is run.

```
(shiftstack) [cloud-user@bastion ~]$ openstack container list -f value

openshift-registry
```

This reference architecture uses Ceph Rados Gateway. When a Cinder persistent volume is used it is attached to a single infrastructure node which can become a single point of failure.

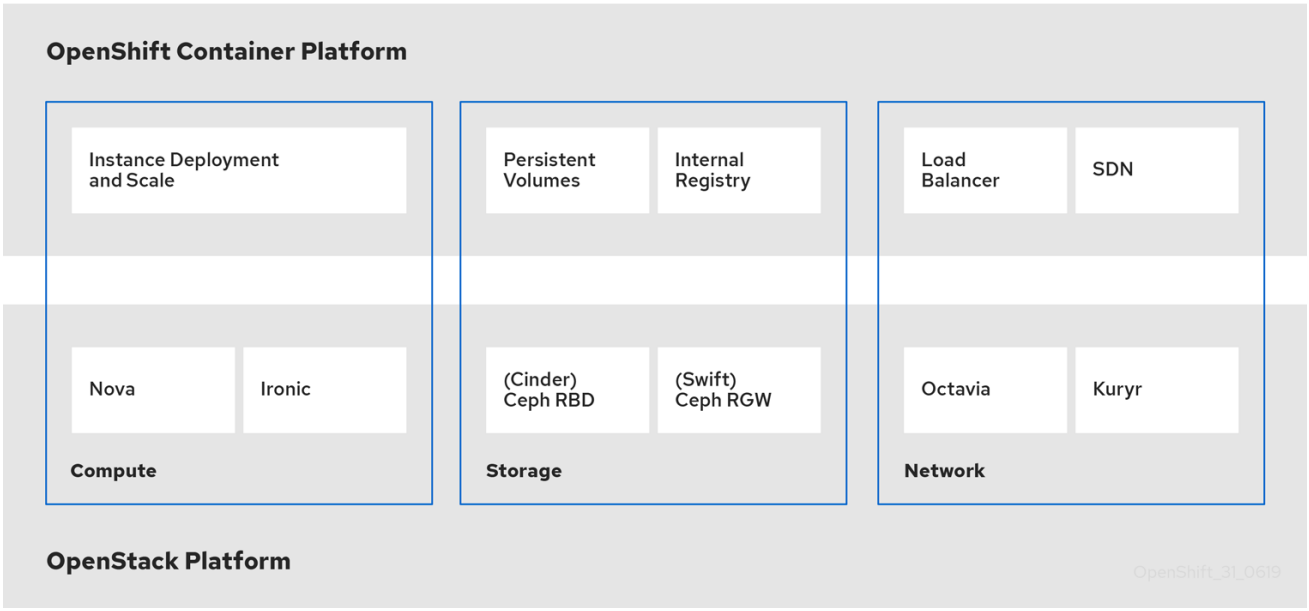
Note that by default the OpenStack user specified to configure Ceph Rados Gateway in **install.yml** must have the **admin** or **Member** Keystone role.

CHAPTER 5. KEY INTEGRATION POINTS

OpenShift Container Platform and Red Hat OpenStack Platform are co-engineered to form a robust, scalable, and secure platform for running self-service virtual machine and containers. This section of the reference architecture describes the key integration points between the products in greater detail.

Figure 8 maps the key integration points from an OpenStack service to the corresponding OpenShift components.

Figure 8:

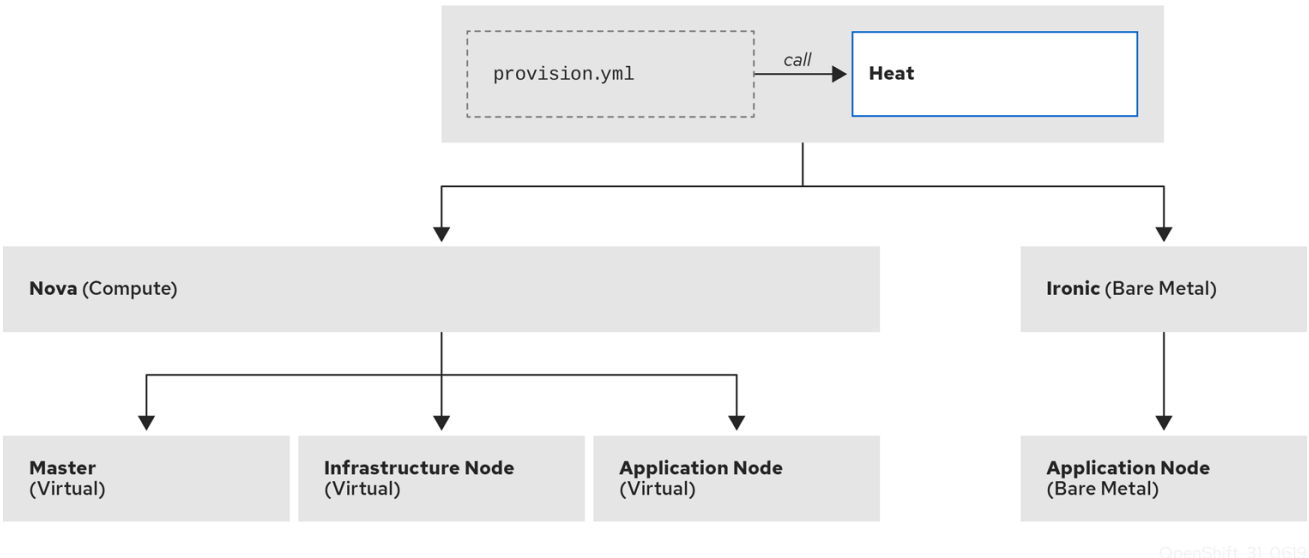


The integrations are divided into three categories: compute, network, and storage.

5.1. COMPUTE

Heat is OpenStack's orchestration service. It can launch composite cloud applications based on text-file templates that can be managed as code. The openshift-ansible installer makes native calls to the Heat API to deploy OpenShift instances, networks, and storage.

Figure 9:



The installer uses the Ansible OpenStack cloud modules to manage OpenStack resources. The Ansible playbook provision.yml makes the calls to Heat. This is depicted in figure 9.

Heat is responsible for orchestrating the Nova Compute service and the Ironic Bare metal service. Nova creates virtual machines for running various OpenShift roles based on predefined flavors and images. Ironic can also be used to push operating system images to bare metal servers that meet or exceed the minimum hardware requirements for the role's flavor.

Once the OpenShift cloud stack is deployed, it can also be scaled using the same facilities.

```
(shiftstack) [cloud-user@bastion ~]$ openstack stack resource list openshift-cluster -c
resource_name -c resource_type -c resource_status

+-------------------------------+-----------------------------+-----------------+

| resource_name | resource_type | resource_status |

+-------------------------------+-----------------------------+-----------------+

| compute_nodes | OS::Heat::ResourceGroup | CREATE_COMPLETE |

| api_lb | OS::Octavia::LoadBalancer | CREATE_COMPLETE |

| router_lb_pool_http | OS::Octavia::Pool | CREATE_COMPLETE |

| common-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |

| service_subnet | OS::Neutron::Subnet | CREATE_COMPLETE |

| api_lb_pool | OS::Octavia::Pool | CREATE_COMPLETE |

| router_lb_floating_ip | OS::Neutron::FloatingIP | CREATE_COMPLETE |

| infra-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |

| cns | OS::Heat::ResourceGroup | CREATE_COMPLETE |

| lb-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |

| infra_nodes | OS::Heat::ResourceGroup | CREATE_COMPLETE |

| router_lb_listener_https | OS::Octavia::Listener | CREATE_COMPLETE |

| router_lb_pool_https | OS::Octavia::Pool | CREATE_COMPLETE |

| etcd-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |

| router_lb | OS::Octavia::LoadBalancer | CREATE_COMPLETE |

| etcd | OS::Heat::ResourceGroup | CREATE_COMPLETE |

| service_net | OS::Neutron::Net | CREATE_COMPLETE |

| pod_subnet | OS::Neutron::Subnet | CREATE_COMPLETE |

| pod_access_sg | OS::Neutron::SecurityGroup | CREATE_COMPLETE |
```

```
| master-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |
| pod_net | OS::Neutron::Net | CREATE_COMPLETE |
| api_lb_listener | OS::Octavia::Listener | CREATE_COMPLETE |
| router_lb_listener_http | OS::Octavia::Listener | CREATE_COMPLETE |
| masters | OS::Heat::ResourceGroup | CREATE_COMPLETE |
| service_router_port | OS::Neutron::Port | CREATE_COMPLETE |
| pod_subnet_interface | OS::Neutron::RouterInterface | CREATE_COMPLETE |
| api_lb_floating_ip | OS::Neutron::FloatingIP | CREATE_COMPLETE |
| internal_api_lb_listener | OS::Octavia::Listener | CREATE_COMPLETE |
| node-secgrp | OS::Neutron::SecurityGroup | CREATE_COMPLETE |
| service_subnet_interface | OS::Neutron::RouterInterface | CREATE_COMPLETE |
+---------------------------------+------------------------------+-----------------+
```

Abstracting API calls to the native OpenStack services has multiple benefits:

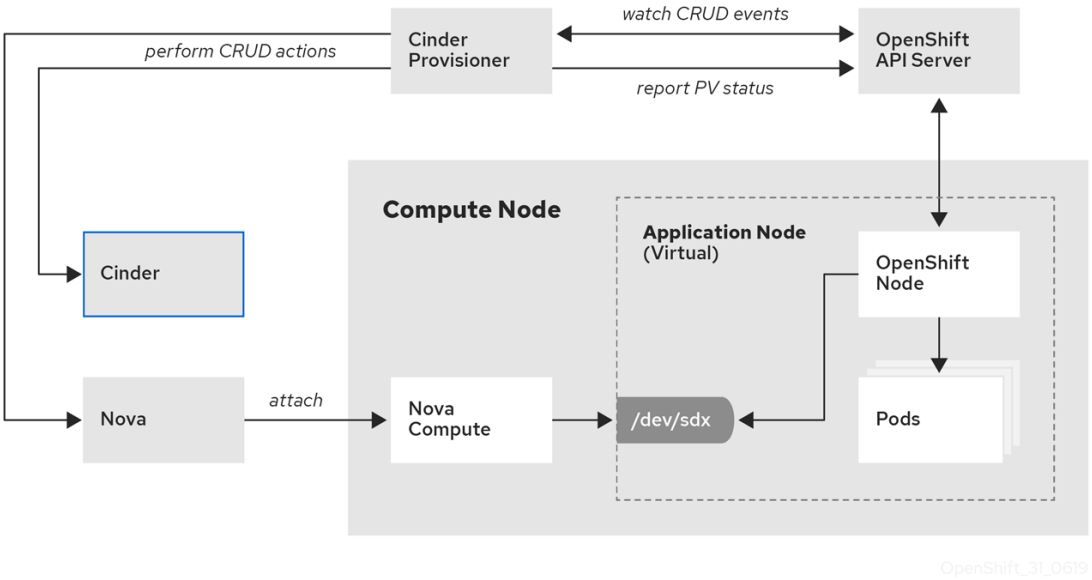
- Administrators do not have to learn Heat or any other OpenStack tools to deploy OpenShift on OpenStack.
- OpenStack administrators familiar with Heat can use the tools they are already familiar with the examine and manage the deployed stack. The preceding code block shows a Heat resource listing from a stack deployed by openshift-ansible.
- Heat provides a scalable and reliable interface for automating OpenShift installations.

5.2. STORAGE

The persistent volume framework allows OpenShift users to request block storage volumes without any knowledge of the underlying storage that is used. The storage must exist in the underlying infrastructure before it can be used in OpenShift.

The openshift-ansible installer configured a Cinder storageClass so that persistent volumes can be dynamically provisioned and attached when they are requested. This is depicted in figure 10.

Figure 10:



OpenShift API events are modeled on actions taken against API objects. The Cinder provisioner monitors the OpenShift API server for CRUD events related to persistent volume requests. When a persistent volume claim is generated by a pod, the Cinder provisioner performs actions to create the Cinder volume.

Once the volume is created, a subsequent call to the Nova API attaches the volume to the Nova compute node where the pod resides. The volume is then attached to the pod. The Cinder provisioner continues monitoring the OpenShift API server for volume release or delete requests, and initiates those actions when they are received.

The openshift-ansible installer also interacts with Cinder to create persistent block storage for the OpenShift instances. The master and node instances contain a volume to storage container images. The purpose of the container volume is to ensure that container images do not compromise node performance or local storage.

5.2.1. Internal registry

The OpenShift image registry requires persistent storage to ensure that images are saved in the event that the registry pod needs to migrate to another node. The openshift-ansible installer can automatically create a Cinder volume for internal registry persistent storage, or it can point to a pre-created volume. Alternately, openshift-ansible can use an S-3 compatible object store to hold container images. Either OpenStack Swift or Ceph Rados Gateway are suitable object stores for the internal container registry.

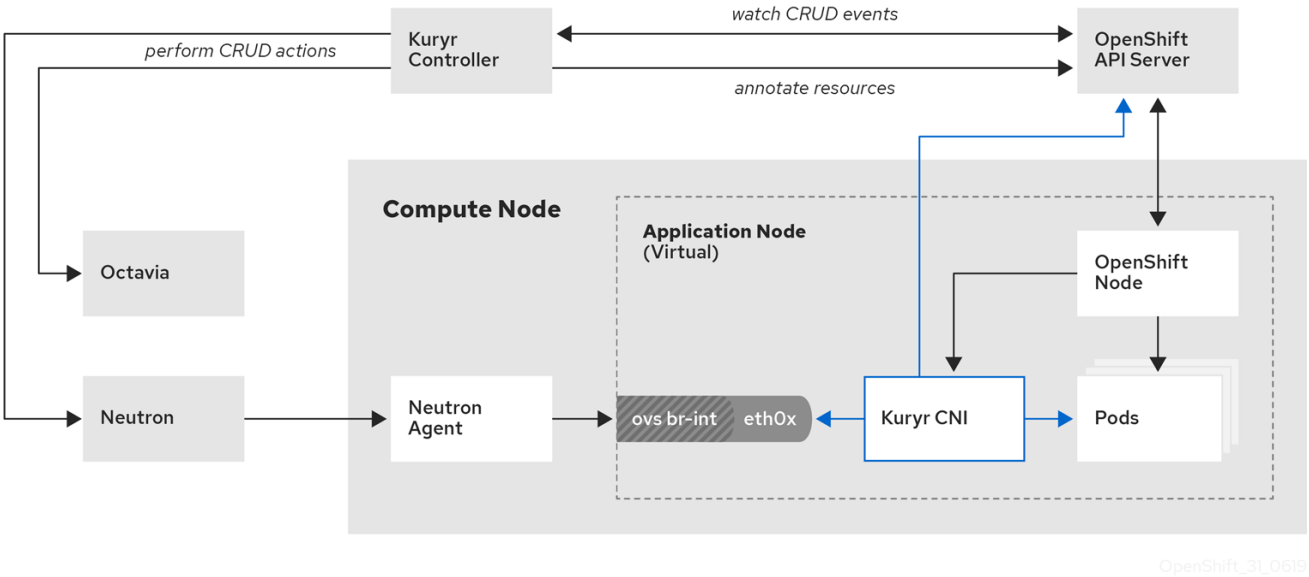
5.3. NETWORK

There are two primary network integration points: Kuryr gives OpenShift pods direct access to OpenStack Neutron networks. Octavia automates load balancer creation and configuration for OpenShift services.

5.3.1. Kuryr

Kuryr is a CNI plugin that uses OpenStack Neutron and Octavia to provide networking for pods and services. It is primarily designed for OpenShift clusters running on OpenStack virtual machines. Figure 11 is a block diagram of a Kuryr architecture.

Figure 11:



Kuryr components are deployed as pods in the kuryr namespace. The kuryr-controller is a single container service pod installed on an infrastructure node as a deployment OpenShift resource type. The kuryr-cni container installs and configures the kuryr CNI driver on each of the OpenShift masters, infrastructure nodes, and compute node as a daemonset.

The Kuryr controller watches the OpenShift API server for pod, service, and namespace create, update, and delete events. It maps the OpenShift API calls to corresponding objects in Neutron and Octavia. This means that every network solution that implements the Neutron trunk port functionality can be used to back OpenShift using Kuryr. This includes open source solutions such as OVS and OVN as well as Neutron-compatible commercial SDNs. That is also the reason the openvswitch firewall driver is a prerequisite instead of the ovs-hybrid firewall driver.

Kuryr is recommended for OpenShift deployments on encapsulated OpenStack tenant networks in order to avoid double encapsulation: running an encapsulated OpenShift SDN over top of an encapsulated OpenStack network, such as whenever VXLAN/GRE/GENEVE are required. Implementing Kuryr does not make sense when provider networks, tenant VLANs, or a third party commercial SDN such as Cisco ACI, or Juniper Contrail is implemented.

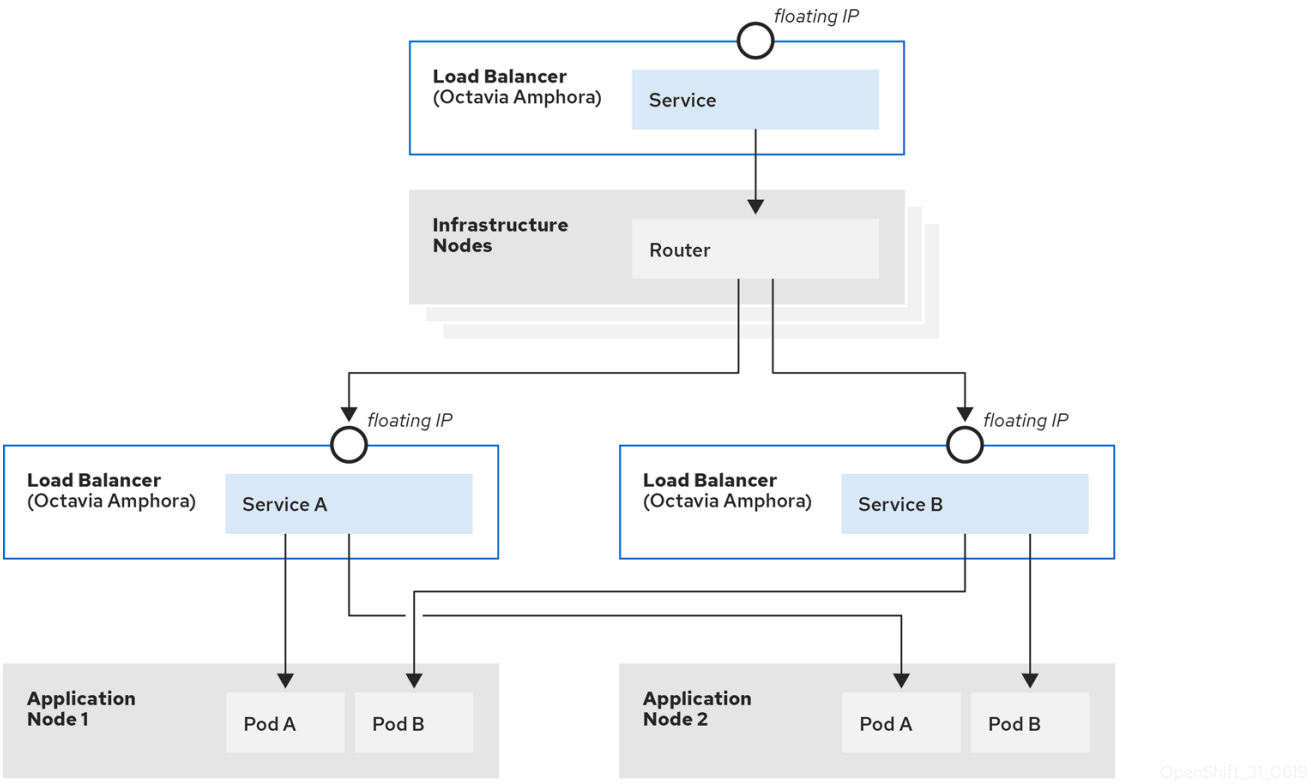
The OpenShift Network Policy SDN is recommended for unencapsulated OpenStack networks such as tenant VLANs or provider networks. Kuryr is also not needed when OpenStack Neutron is backed by a commercial SDN with hardware acceleration.

More information on OpenShift Container Platform 3.11 SDN can be found in the Configuring the SDN section of the official documentation.

5.3.2. Octavia

Octavia is an operator-grade, open source, scalable load balancer. It implements load balancing functionality by launching virtual machine appliances (called amphora) in the OpenStack service project. The amphora run HAproxy services. Figure 12 depicts the Octavia architecture and its relationship to OpenShift.

Figure 12:



The load balancer is Octavia's front end service. Each load balancer has a listener that maps a virtual IP address and port combination to a pool of pods. The openshift-ansible installer configures Octavia load balancers for the API server, cluster routers, and registry access.

```
(shiftstack) [stack@undercloud ~]$ openstack loadbalancer list -f table -c name -c vip_address -c provider

+----------------------------------------------+---------------+----------+

| name | vip_address | provider |

+----------------------------------------------+---------------+----------+

| openshift-ansible-openshift.example.io-api-lb | 172.30.0.1 | octavia |

| openshift-cluster-router_lb-gc2spptdjh46 | 172.16.1.5 | octavia |

| default/docker-registry | 172.30.95.176 | octavia |

| default/registry-console | 172.30.164.5 | octavia |

| default/router | 172.30.220.98 | octavia |

| openshift-monitoring/grafana | 172.30.242.217 | octavia |

| openshift-web-console/webconsole | 172.30.10.9 | octavia |

| openshift-monitoring/prometheus-k8s | 172.30.134.157 | octavia |

| openshift-console/console | 172.30.235.94 | octavia |
```

```
| openshift-metrics-server/metrics-server | 172.30.237.31 | octavia |

| openshift-monitoring/alertmanager-main | 172.30.189.57 | octavia |

| openshift-logging/logging-kibana | 172.30.226.107 | octavia |

| openshift-infra/hawkular-cassandra | 172.30.207.19 | octavia |

+------------------------------------------------+---------------+----------+
```

This command output shows the OpenStack Octavia load balancers that correspond to the underlying OpenShift services. Notice that all VIP addresses belong to the OpenShift service network except for the cluster router load balancer, which has a VIP on the deployment network.

Octavia is also used by OpenShift to load balance across pod replica sets. When an OpenShift service is exposed as a LoadBalancer type, an Octavia load balancer is automatically created to load balance client connections to the service pods. This also happens when creating a ClusterIP service. Namespace isolation is enforced when kuryr ClusterIP services are created. It is not enforced when LoadBalancer services are created because they should be externally accessible. A floating IP is added to the load balancer VIP port.

CHAPTER 6. SOLUTION VALIDATION

The following feature validation tests were performed by Red Hat engineers in internal lab environments:

- Kuryr networking, including namespace isolation and port pre-creation.
- Automatic floating IP and load balancer assignment when exposing OpenShift service through a load balancer.
- OVS-multitenant SDN networking.
- OpenShift namespace isolation with network policy.
- Deploying all OpenStack instances with floating IP addresses (no deployment host).
- Deploying all OpenStack instances on an isolated network using a deployment host.
- Internal Neutron DNS + OpenStack local DNS forwarding.
- Hybrid internal Neutron DNS + external public DNS records using nssupdate.
- Bring your own DNS using tenant-internal BIND server and nssupdate.
- Bring your own DNS using external BIND server and nssupdate.
- Bring your own DNS using external BIND server and manual DNS record precreation.
- OpenStack public-endpoint SSL.
- Ceph RGW internal registry.
- Barbican encryption of Ceph-RGW internal registry store.
- Cinder-attached internal registry.
- Ceph RBD persistent volume claims.
- Ceph RBD-backed persistent Nova instances.
- Ephemeral Nova instances.
- Colocated etcd.
- Non-HA OpenShift.
- HA OpenShift.
- HA OpenShift plus soft instance affinity.
- Exposing Nvidia GPU through OpenShift on OpenStack using PCI Passthrough.

CHAPTER 7. SUMMARY

To summarize, organizations across the globe are looking to rapidly develop innovative software applications, and a lot of these applications have to be deployed in an on-premises private cloud or in a co-location facility for various reasons (for example, security and compliance, data affinity, performance, among others). The IT organizations responsible for operating the private cloud desire it to be simple, agile, flexible, secure, cost efficient, and be a part of their overall Hybrid and Multi cloud architecture.

This Reference Architecture showcased a prescriptive and pre-validated private cloud solution from Red Hat that allows you to run IT as a Service (ITaaS), and provides rapid provisioning and lifecycle management of containerized apps, virtual machines (VMs), and associated application and infrastructure services. Red Hat OpenShift Container Platform, Red Hat OpenStack Platform, and Red Hat Ceph Storage are the key architectural components of this solution. It can be easily extended to Hybrid and Multi-Cloud with OpenShift Container Platform serving as the common container and kubernetes platform across all clouds. As noted in the previous section, extensive lab testing validated the solution, so that organizations can quickly deploy it and achieve fast time to value by eliminating time required to develop best practices.

CHAPTER 8. ACKNOWLEDGEMENTS

Jacob Liberman	Technical content and test
Abhinav Joshi	Content
Brent Roskos	Technical review
August Simonelli	Technical review and test
Luis Tomas Bolivar	Engineering review – kuryr
Carlos Goncalves	Engineering review – octavia
Sean Cohen	Technical review – storage
Ramon Acedo	Technical review
Derek Cadzow	Documentation review
Martin Lopes	Documentation review
Mark Schmitt	Content & technical review
Rosa Guntrip	Content review
Stephane Lefrere	Content review

CHAPTER 9. APPENDIX A: POD PLACEMENT BY ROLE

9.1. MASTER POD PLACEMENT

Master Pod Placement	
Namespace	**Pod**
kube-system	master-api
kube-system	master-controllers
kube-system	master-etcd
kuryr	kuryr-cni
openshift-console	console
openshift-logging	logging-fluentd
openshift-monitoring	node-exporter
openshift-node	sync
openshift-web-console	webconsole

9.2. INFRASTRUCTURE NODE POD PLACEMENT

Infrastructure Node Pod Placement	
Namespace	**Pod**
kuryr	kuryr-controller
kuryr	kuryr-cni
openshift-infra	hawkular-metrics
openshift-infra	heapster
openshift-logging	logging-es-data-master
openshift-logging	logging-fluentd
openshift-logging	logging-kibana

Infrastructure Node Pod Placement	
openshift-logging	logging-curator
openshift-logging	logging-es-data-master
openshift-monitoring	alertmanager
openshift-monitoring	cluster-monitoring-operator
openshift-monitoring	grafana
openshift-monitoring	node-exporter
openshift-monitoring	prometheus
openshift-node	sync

9.3. APPLICATION NODE POD PLACEMENT

Application Node Pod Placement	
Namespace	**Pod**
kuryr	kuryr-cni
openshift-logging	logging-fluentd
openshift-monitoring	node-exporter
openshift-node	sync

CHAPTER 10. REFERENCES

- OpenShift Container Platform 3.11: Configuring for OpenStack: https://docs.openshift.com/container-platform/3.11/install_config/configuring_openstack.html
- OpenShift Container Platform 3.11 : Kubernetes Infrastructure: https://docs.openshift.com/container-platform/3.11/architecture/infrastructure_components/kubernetes_infrastructure.html
- Ansible OpenStack Cloud module: https://docs.ansible.com/ansible/latest/modules/list_of_cloud_modules.html#openstack
- OpenShift Container Platform 3.11: Internal Registry Overview: https://docs.openshift.com/container-platform/3.11/install_config/registry/index.html
- OpenShift Container Platform 3.11: High Availability Masters: https://docs.openshift.com/container-platform/3.11/architecture/infrastructure_components/kubernetes_infrastructure.html#high-availability-masters
- Red Hat OpenStack Platform 13 Security and Hardening Guide: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/security_and_hardening_guide/index
- Red Hat OpenStack Platform 13 Director Installation and Usage Guide: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/director_installation_and_usage/index
- OpenShift Platform 3.11 Documentation: Planning your Installation: https://docs.openshift.com/container-platform/3.11/install/index.html
- OpenShift Enterprise 3.0 Documentation: SDN Concepts: https://docs.openshift.com/enterprise/3.0/architecture/additional_concepts/sdn.html#architectu additional-concepts-sdn
- Red Hat Ceph Storage Hardware Selection Guide: https://www.redhat.com/en/resources/resources-red-hat-ceph-storage-hardware-selection-guide-html
- Red Hat OpenStack Platform 13 Documentation: Installing Barbican: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/manage_secrets_with_openstack_key_manager/installinc
- OpenShift Container Platform 3.11: Persistent Storage: https://docs.openshift.com/container-platform/3.11/architecture/additional_concepts/storage.html#pv-access-modes
- OpenShift Container Platform 3.11: Configuring the SDN: https://docs.openshift.com/container-platform/3.11/install_config/configuring_sdn.html