Executive Summary

The term “cloud computing” is often associated with virtual machines, but many emerging and rapidly growing cloud technologies now make use of containerization in place of or alongside virtualization. Occupying a much smaller footprint than virtual machines, containers can run multiple isolated applications on a host. Container-based virtualization offers many benefits when compared to traditional virtualization technologies, and containers are perceived as a faster, more portable way to deploy services on cloud infrastructures.

While containers themselves provide many benefits, it can be a challenge to manage large containerized environments. That’s why many container orchestration tools have gained popularity. Each orchestration tool is different, however, and should be chosen individually for specific purposes.

Containers can be deployed and terminated with a minimum overhead in seconds. However, when data persistence is important, their ephemeral nature may create a challenge. The next frontier of containerization will be reliable and efficient connection to storage.

This reference architecture (RA) will show you how to prepare, provision, deploy, and manage a Red Hat® OpenShift Container Platform 3.6–based private cloud environment with additional container-native storage solutions. The intended audience for this RA are system administrators or system architects. Some experience with Docker* and OpenShift technologies might be helpful, but is not required.

Intel, Red Hat, and OpenShift

Hardware Summary

- **Intel® Server System R1208WFTYS** which are 1U, two-socket servers optimized for a cloud/data center market.
- **Intel® Server System R2224WFTZS** which are 2U, two-socket servers optimized for a cloud/data center and storage market.
- **Two (2) Arista™ DCS-7060CX-32S** which are data plane, high performance switches.
- **Enterprise-class L2 management switch**

Software Summary

- **Red Hat Enterprise Linux® and Red Hat Enterprise Linux Atomic Host** operating systems installed on Red Hat OpenShift cluster nodes as a standard platform for hosting and managing the environment.
- **Red Hat OpenShift Container Platform** adds developer- and operation-centric tools to enable rapid application development, easy deployment, scaling, and long-term lifecycle maintenance for small and large teams and applications.
- **Red Hat container-native storage** is a set of tools that provides persistent storage for containers. It includes Red Hat Gluster* Storage as a storage backend and Heketi which manages Red Hat Gluster Storage and serves an API for its configuration.
- **iPXE Environment** is a set of containerized tools for automated bare-metal provisioning.
Furthermore, Red Hat OpenShift Container Platform architecture makes use of the following software:

- **Docker** to build, ship, and run containerized applications
- **Kubernetes** to orchestrate and manage containerized applications
- **Etcd***, which is a key-value store for the OpenShift Container Platform cluster
- **Open vSwitch*** to provide software-defined networking (SDN)-specific functions in the OpenShift Container Platform environment
- **Red Hat Ansible*** Automation for installation and management of the OpenShift Container Platform deployment
- **HAProxy*** for routing and load-balancing purposes
- **Keepalived*** for virtual IP management for HAProxy instances

All software components were installed using the versions shown in Table 1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>7.4</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux Atomic Host</td>
<td>7.4.0</td>
</tr>
<tr>
<td>Red Hat OpenShift Container Platform</td>
<td>3.6</td>
</tr>
<tr>
<td>Red Hat container-native storage</td>
<td>3</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>1.6.1</td>
</tr>
<tr>
<td>Docker</td>
<td>1.12.6</td>
</tr>
<tr>
<td>Ansible</td>
<td>2.3.2.0</td>
</tr>
<tr>
<td>Etcd</td>
<td>3.1.9</td>
</tr>
<tr>
<td>Open vSwitch</td>
<td>2.7.2</td>
</tr>
<tr>
<td>HAPerxy</td>
<td>1.5.18</td>
</tr>
<tr>
<td>Keepalived</td>
<td>1.3.5</td>
</tr>
<tr>
<td>Red Hat Gluster Storage</td>
<td>3.3.0</td>
</tr>
<tr>
<td>GlusterFS</td>
<td>3.8.4</td>
</tr>
<tr>
<td>Heketi</td>
<td>4.0.0</td>
</tr>
</tbody>
</table>
Red Hat Enterprise Linux

This RA uses two types of operating systems—Red Hat Enterprise Linux 7.4 (for the bastion node) and Red Hat Enterprise Linux Atomic Host 7.4.0 (for master, infrastructure, storage, and application nodes).

The Red Hat Enterprise Linux operating system helps ensure high quality and a reliable base for the whole system, provides strong security features, supports business-critical workloads, and is interoperable with a variety of operating systems.

Red Hat Enterprise Linux Atomic Host is a lightweight variant of the Red Hat Enterprise Linux operating system designed to run Linux containers. By combining the modular capabilities of Linux containers and Red Hat Enterprise Linux, containers can be more securely and effectively deployed and managed across public, private, and hybrid cloud infrastructures.

Red Hat Ansible* Automation

Ansible is a powerful IT automation tool. It is capable of provisioning numerous types of resources and deploying applications. It can configure and manage devices and operating system components. Due to the simplicity, extensibility, and portability, this OpenShift RA is based largely on Ansible playbooks. That is also the reason why this RA adopts and distributes improvements in the same manner.

Red Hat Container-Native Storage

Red Hat container-native storage makes OpenShift Container Platform a fully hyper converged infrastructure where storage containers co-reside with the compute containers. Storage plane is based on containerized Red Hat Gluster Storage services, which controls storage devices on every storage server. Heketi is a part of the container-native storage architecture and controls all of the nodes that are members of storage cluster. Heketi also provides an API through which storage space for containers can be easily requested. While Heketi provides an endpoint for storage cluster, the object that makes calls to its API from OpenShift clients is called a Storage Class. It is a Kubernetes and OpenShift object that describes the type of storage available for the cluster and can dynamically send storage requests when a persistent volume claim is generated. Figure 1 describes a basic flow of persistent volume creation in container-native storage architectures.

Figure 1. Persistent volume request in container-native storage

This Red Hat OpenShift RA contains five types of nodes: bastion, master, infrastructure, storage, and application, which are described below.

Bastion Node

This is a dedicated node that serves as the main deployment and management server for the Red Hat OpenShift cluster. It is used as the logon node for the cluster administrators to perform the system deployment and management operations, such as running the Ansible OpenShift deployment playbooks. The bastion node runs Red Hat Enterprise Linux 7.4.

OpenShift Master Nodes

The OpenShift Container Platform master is a server that performs control functions for the whole cluster environment. It is responsible for the creation, scheduling, and management of all objects specific to Red Hat OpenShift. It includes API, controller manager, and scheduler capabilities in one OpenShift binary. It is also a common practice to install an etcd key-value store on OpenShift masters to achieve a low-latency link between etcd and OpenShift masters. It is recommended that you run both Red Hat OpenShift masters and etcd in highly available environments. This can be achieved by running multiple OpenShift masters in conjunction with an external active-passive load balancer and the clustering functions of etcd. The OpenShift master node runs Red Hat Enterprise Linux Atomic Host 7.4.0.

Red Hat OpenShift System Architecture

The Red Hat OpenShift Container Platform is a complete container application platform that includes the application development process in one consistent solution across multiple infrastructure footprints. Red Hat OpenShift integrates the architecture, processes, platforms, and services needed to help development and operations teams traverse traditional siloed structures and produce applications that help businesses succeed.

The Red Hat OpenShift cluster platform is managed by the Kubernetes container orchestrator, which manages containerized applications across a cluster of systems running the Docker container runtime environment. The physical configuration of the Red Hat OpenShift platform is based on the Kubernetes cluster architecture.
OpenShift Infrastructure Nodes

The OpenShift infrastructure node runs infrastructure-specific services: Docker Registry*, HAProxy router, and Heketi. Docker Registry stores application images in the form of containers. The HAProxy router provides routing functions for Red Hat OpenShift applications. It currently supports HTTP(S) traffic and TLS-enabled traffic via Server Name Indication (SNI). Heketi provides management API for configuring GlusterFS persistent storage. Additional applications and services can be deployed on OpenShift infrastructure nodes. The OpenShift infrastructure node runs Red Hat Enterprise Linux Atomic Host 7.4.0.

OpenShift Application Nodes

The OpenShift application nodes run containerized applications created and deployed by developers. An OpenShift application node contains the OpenShift node components combined into a single binary, which can be used by OpenShift masters to schedule and control containers. A Red Hat OpenShift application node runs Red Hat Enterprise Linux Atomic Host 7.4.0.

OpenShift Storage Nodes

The OpenShift storage nodes run containerized GlusterFS services which configure persistent volumes for application containers that require data persistence. Persistent volumes may be created manually by a cluster administrator or automatically by storage class objects. An OpenShift storage node is also capable of running containerized applications. A Red Hat OpenShift storage node runs Red Hat Enterprise Linux Atomic Host 7.4.0.

Figure 2. OpenShift Cluster Roles
Reference Architecture | Deploying Red Hat OpenShift® Container Platform 3.6 with Container-Native Storage

Hardware Detail

The Red Hat OpenShift RA is validated using Intel® servers and Arista® network switches. The configuration includes six application nodes, three master nodes, two infrastructure nodes, three storage nodes, and one bastion node. 

**Figure 3** shows the rack-level diagram of the hardware. In addition to the servers and switches, the rack also includes power distribution units (PDUs) and the necessary cables for management and data connectivity across the servers and switches. The full bill of materials required to implement this infrastructure is provided in Table 3 of the Hardware Configuration section of this document.

**Figure 4. Intel® S2600WF family server board**

Intel® Server System S2600WF Family

The Intel® Server Board S2600WF product family delivers power and performance at peak efficiency in 1U and 2U rack mount server form factors that feature energy-efficient dual Intel® Xeon® Scalable processors. High memory capacity, networking, storage, and I/O flexibility combine with innovative design to provide an exceptional, reliable server for business IT, appliance, data center, cloud, and high performance computing applications.

This reference architecture uses two types of Intel® Server Systems: R1208WFTYS and R2224WFTZS.

**Figure 5. Intel® Server System R1208WFTYS**

Intel® Server System R1208WFTYS is a 1U rack server based on Intel® Server Board S2600WFT, which is a dual processor board optimized for the cloud and data center market. It is compatible with Intel® Xeon® Scalable processors, supporting up to 24 DIMMs, eight 2.5” hot-swap front drives, two M2.SSD internal drives, and two 10Gb Ethernet ports. It includes an Intel® OCP Module for adding additional features without losing a PCIe* add-in slot. The R1208WFTYS can be configured with a redundant power supply.

**Figure 6. Intel® Server System R2224WFTZS**

Intel® Server System R2224WFTZS is a 2U rack server based on the same Intel® Server Board S2600WFT. Also using the Intel® Xeon® Scalable processor series, the R2224WFTZS supports up to 24 DIMMs, twenty-four 2.5” hot-swap front drives, four M2.SSD internal drives, and two 10Gb Ethernet ports. It may be configured with a redundant power supply.
Intel® Xeon® Scalable Processor Family

Intel® Xeon® Scalable processors provide a new foundation for secure, agile, multi-cloud data centers. This platform provides businesses with breakthrough performance to handle system demands ranging from entry-level cloud servers to compute-hungry tasks including real-time analytics, virtualized infrastructure, and high performance computing. This processor family includes technologies for accelerating and securing specific workloads.

- **Intel® Advanced Vector Extensions 512 (Intel® AVX-512)** expands the number of registers and adds instructions to streamline processing. The result is higher performance of mixed workloads.

- **Intel® Transactional Synchronization Extensions New Instructions (Intel® TSX-NI)** provide higher performance of multi-threaded workloads.

- **Intel® Trusted Execution Technology (Intel® TXT)** provides the necessary underpinnings to evaluate the computing platform and its security.

- **Intel® Platform Trust Technology (Intel® PTT)** provides secure encryption keys in a Trusted Platform Module (TPM) integrated directly into the chipset.

Intel® Xeon® Scalable processors are now available in four feature configurations:

- **Intel® Xeon® Bronze processors** provide affordable performance for small business and basic storage.

- **Intel® Xeon® Silver processors** provide essential performance and power efficiency.

- **Intel® Xeon® Gold processors** provide workload-optimized performance and advanced reliability.

- **Intel® Xeon® Platinum processors** provide demanding, mission-critical performance for AI, analytics, and hybrid cloud workloads.

Intel® Ethernet Network Adapter XXV710

Intel® Ethernet Network Adapter XXV710 delivers excellent performance with a theoretical throughput of 50Gb/s (25Gb/s single-port bi-directional), in a PCI Express® v3.0 x8 slot. The Intel® Ethernet Network Adapter XXV710 is based on an innovative new architecture, with its ability to auto-negotiate for 1/10/25GbE connections, and is designed to meet the needs of customers who have multiple speeds deployed in their environment.

Optimized performance vectors and key uses include:

- **Small Packet Performance**: Achieves wire-rate throughput on smaller payload sizes

- **Bulk Transfer Performance**: Delivers line-rate performance with low CPU usage for large application buffers

- **Virtualized Performance**: Alleviates hypervisor I/O bottlenecks by providing flow separation for Virtual Machines

- **Network Virtualization**: Offloads network virtualization overlays including VXLAN, NVGRE, GENEVE, MPLS, VXLAN-GPE with NSH

Intel® SSD DC S4500 Series

Intel® SSD DC S4500 Series is a storage-inspired SATA SSD optimized for read-intensive workloads. Based on TLC Intel® 3D NAND Technology, these larger-capacity SSDs enable data centers to increase data stored per rack unit. The Intel® SSD DC S4500 Series is built for compatibility with legacy infrastructures, enabling easy storage upgrades that minimize the costs associated with modernizing data centers. This 2.5” 7mm form factor offers a wide range of capacities from 240GB up to 3.8TB.
Intel® SSD DC P4600 Series

Packed with a deep feature set, this Intel® 3D NAND SSD for data centers is optimized for the data caching needs of cloud storage and software-defined infrastructures. It modernizes the data center with a combination of performance, capacity, manageability, and reliability. The Intel® SSD DC P4600 Series significantly increases server agility and utilization—while also accelerating applications—across a wide range of cloud workloads. This PCIe NVMe® SSD series comes in two configuration options with a 4TB capacity.

Arista* DCS-7060CX-32S Switch

The Arista 7060X and 7260X Series are purpose-built 10, 25, 40, 50, and 100GbE data center switches in compact and energy efficient form factors with wirespeed layer 2 and layer 3 features, combined with advanced features for software defined cloud networking.

Featuring 32 QSFP100 and 2 SFP+ ports in a 1RU form factor the 7060CX-32S switch delivers feature-rich layer 2 and layer 3 wirespeed performance with an overall throughput of 6.4Tbps. For configuration flexibility, the 7060CX-32S supports up to 32x 100GbE ports, where each port can be broken out into wide range of speed choices: 4x 10GbE, 4x 25GbE, 1x 40GbE, or 2x 50GbE, in addition to 1x 100GbE.

Key high availability features include:
- 1+1 hot-swappable power supplies and four N+1 hot-swap fans
- Color coded PSUs and fans
- Live software patching
- Self-healing software with Stateful Fault Repair (SFR)
- Smart System Upgrade (SSU) and Accelerated Software Update (ASU)
- Up to 64 10/25/40/50/100GbE ports per link aggregation group
- Multi-chassis LAG for active/active L2 multi-pathing
- 128-way ECMP routing for load balancing and redundancy

The Arista 7060X and 7260X series deliver line rate switching at layer 2 and layer 3 to enable faster and simpler network designs for data centers that dramatically lowers the network capital and operational expenses. Visit the product page at www.arista.com/en/products/7060x-series for more details.
## Hardware Configuration

**Table 2** summarizes the configuration of the various node types in the Red Hat OpenShift cluster in this RA.

### Table 2. OpenShift node roles and hardware configuration

<table>
<thead>
<tr>
<th>Red Hat OpenShift® role</th>
<th>Qty.</th>
<th>Platform</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| Bastion node           | 1    | 1U Intel® Server System S2600WF | • 2x Intel® Xeon® Silver 4114 processor at 2.20GHz  
• 192GB memory (12x 16GB)  
• 2x Intel® SSD DC S4500 Series 240GB enterprise entry SATA G3HS 2.5” SSDs (RAID1)  
• 1x Intel® RAID Module RMSP3HD080E  
• 1x Intel® Remote Management Module 4 Lite 2 AXXRMM4LITE2  
• 1x Intel® Ethernet Network Adapter XXV710-DA2, dual-port 25Gbps SFP28 |
| Master node            | 3    | 1U Intel® Server System S2600WF | • 2x Intel® Xeon® Silver 4114 processor at 2.20GHz  
• 192GB memory (12x 16GB)  
• 2x Intel® SSD DC S4500 Series 240GB enterprise entry SATA G3HS 2.5” SSDs (RAID1)  
• 1x Intel® RAID Module RMSP3HD080E  
• 1x Intel® Remote Management Module 4 Lite 2 AXXRMM4LITE2  
• 1x Intel® Ethernet Network Adapter XXV710-DA2, dual-port 25Gbps SFP28 |
| Infrastructure node    | 2    | 1U Intel® Server System S2600WF | • 2x Intel® Xeon® Silver 4114 processor at 2.20GHz  
• 192GB memory (12x 16GB)  
• 2x Intel® SSD DC S4500 Series 240GB enterprise entry SATA G3HS 2.5” SSDs (RAID1)  
• 1x Intel® RAID Module RMSP3HD080E  
• 1x Intel® Remote Management Module 4 Lite 2 AXXRMM4LITE2  
• 1x Intel® Ethernet Network Adapter XXV710-DA2, dual-port 25Gbps SFP28 |
| App node               | 6    | 1U Intel® Server System S2600WF | • 2x Intel® Xeon® Silver 4114 processor at 2.20GHz  
• 384GB memory (12x 32GB)  
• 2x Intel® SSD DC S4500 Series 240GB enterprise entry SATA G3HS 2.5” SSDs (RAID1)  
• 1x Intel® SSD DC P4500 Series 4TB PCIe NVMe 2.5” SSDs  
• 1x Intel® RAID Module RSP3WD080E  
• 1x Intel® Remote Management Module 4 Lite 2 AXXRMM4LITE2  
• 1x Intel® Ethernet Network Adapter XXV710-DA2, dual-port 25Gbps SFP28 |
| Storage node           | 3    | 2U Intel® Server System S2600WF | • 2x Intel® Xeon® Silver 4114 processor at 2.20GHz  
• 192GB memory (12x 16GB)  
• 2x Intel® SSD DC S4500 Series 240GB enterprise entry SATA G3HS 2.5” SSDs (RAID1)  
• 20x Intel® SSD DC S4500 Series 3.8TB enterprise entry SATA G3HS 2.5” SSDs  
• 1x Intel® RAID Module RMSP3HD080E  
• 1x Intel® Storage Expander RES3TV360  
• 1x Intel® Remote Management Module 4 Lite 2 AXXRMM4LITE2  
• 1x Intel® Ethernet Network Adapter XXV710-DA2, dual-port 25Gbps SFP28 |

In addition to the servers, the other rack infrastructure components required are listed in **Table 3**.

### Table 3. The Lenovo and Red Hat OpenShift reference architecture hardware bill of materials

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rack enclosure</td>
<td>42U 1,200 mm deep generic rack</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Management switch</td>
<td>Generic</td>
<td></td>
</tr>
<tr>
<td>2–4</td>
<td>Rack power distribution units (PDUs)</td>
<td>Generic</td>
<td></td>
</tr>
</tbody>
</table>
Power Configuration

The recommended power connectivity is shown in Figure 13. The diagram shows the power connectivity for the full rack with 15 Intel® Server System S2600WF nodes and three switches with redundant power supplies. Each power domain contains one PDU.

![Figure 13. Rack power diagram](image)

Networking Overview

The Red Hat OpenShift Container Platform reference architecture uses the 25GbE network as the primary fabric for inter-node communication. As described previously, two Arista DCS-7060CX-32S switches are part of the solution to provide data layer communication. One management switch is used for “out-of-band” communication. Figure 14 shows the network architecture for this solution.

![Figure 14. Active – backup bond configuration](image)
The solution in this reference architecture is designed to deliver maximum availability. Each of the cluster nodes has a link to each peer switch for redundancy. This provides improved high availability (HA) for the nodes using a link failover. Connection to the uplink core network is facilitated by the MLAG peers, which present a logical switch to the uplink network, enabling connectivity with all links active and without a hard requirement for spanning-tree protocol (STP). The link between the two MLAG peers is an inter-switch link (ISL) and provides excellent support of east-west cluster traffic the nodes. The MLAG presents a flexible basis for interconnecting to the uplink/core network, ensures the active usage of all available links, and provides high availability in case of a switch failure or a required maintenance outage.

Network Architecture

There are three logical networks in this RA:

- **External**: The external network is used for the public API, the Red Hat OpenShift web interface, and exposed applications (services and routes).
- **Internal**: This is the primary network used for cluster management and inter-node communication. The same network acts as the layer for server provisioning using PXE and HTTP. Domain Name Servers (DNS) and Dynamic Host Configuration Protocol (DHCP) services also reside on this network to provide the functionality necessary for the deployment process and the cluster to work. Communication with the Internet is provided by NAT configured on the bastion node. Routing between the internal network and the out-of-band network must be provided so that the bastion node can reach other intelligent platform management interface (IPMI) of other nodes.
- **Out-of-band/IPMI**: This is a secured and isolated network used for switch and server hardware management, such as access to the IMM module and SoL (Serial-over-LAN).

![Figure 15. Red Hat OpenShift logical/physical network connectivity](image)
Figure 16 shows the components of Red Hat OpenShift Container Platform and its logical architecture. All Red Hat OpenShift nodes are connected via the internal network, where they can communicate with each other. Furthermore, Open vSwitch creates its own network for Red Hat OpenShift pod-to-pod communication. Because of the multi-tenant plugin, Open vSwitch pods can communicate to each other only if they share the same project namespace. There is a virtual IP address managed by Keepalived on two infrastructure hosts for external access to the Red Hat OpenShift web console and applications. Storage nodes are also connected via internal network as a highly available and fast Gluster Storage cluster.
Network Addresses

Table 4 shows the network and subnet addresses for the various networks in the OpenShift cluster.

<table>
<thead>
<tr>
<th>Network</th>
<th>Network Purpose</th>
<th>VLAN ID</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.25.0/24</td>
<td>IPMI network</td>
<td>-</td>
<td>In-band interface</td>
</tr>
<tr>
<td>172.30.4.0/24</td>
<td>Internal network</td>
<td>3073 (native)</td>
<td>bond0</td>
</tr>
<tr>
<td>100.65.0.0/22</td>
<td>External network</td>
<td>2017</td>
<td></td>
</tr>
</tbody>
</table>

Switch Management Addresses

Table 5 lists the management IP addresses for the switches.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arista* DCS-7060CX-32S</td>
<td>192.168.25.2</td>
</tr>
<tr>
<td>Arista* DCS-7060CX-32S</td>
<td>192.168.25.3</td>
</tr>
</tbody>
</table>

Server Addresses

<table>
<thead>
<tr>
<th>Host name</th>
<th>bond0</th>
<th>bond0.2017</th>
<th>In-bound IMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>172.30.4.10</td>
<td>100.64.80.10</td>
<td>192.168.25.11</td>
</tr>
<tr>
<td>master1</td>
<td>172.30.4.11</td>
<td></td>
<td>192.168.25.12</td>
</tr>
<tr>
<td>master2</td>
<td>172.30.4.12</td>
<td></td>
<td>192.168.25.13</td>
</tr>
<tr>
<td>master3</td>
<td>172.30.4.13</td>
<td></td>
<td>192.168.25.14</td>
</tr>
<tr>
<td>infra1</td>
<td>172.30.4.14</td>
<td>100.65.0.14</td>
<td>192.168.25.15</td>
</tr>
<tr>
<td>infra2</td>
<td>172.30.4.15</td>
<td>100.64.0.15</td>
<td>192.168.25.16</td>
</tr>
<tr>
<td>app1</td>
<td>172.30.4.16</td>
<td></td>
<td>192.168.25.17</td>
</tr>
<tr>
<td>app2</td>
<td>172.30.4.17</td>
<td></td>
<td>192.168.25.18</td>
</tr>
<tr>
<td>app3</td>
<td>172.30.4.18</td>
<td></td>
<td>192.168.25.19</td>
</tr>
<tr>
<td>app4</td>
<td>172.30.4.19</td>
<td></td>
<td>192.168.25.20</td>
</tr>
<tr>
<td>app5</td>
<td>172.30.4.20</td>
<td></td>
<td>192.168.25.21</td>
</tr>
<tr>
<td>app6</td>
<td>172.30.4.21</td>
<td></td>
<td>192.168.25.22</td>
</tr>
<tr>
<td>storage1</td>
<td>172.30.4.22</td>
<td></td>
<td>192.168.25.23</td>
</tr>
<tr>
<td>storage2</td>
<td>172.30.4.23</td>
<td></td>
<td>192.168.25.24</td>
</tr>
<tr>
<td>storage3</td>
<td>172.30.4.24</td>
<td></td>
<td>192.168.25.25</td>
</tr>
</tbody>
</table>

Management Network

A 1Gbps management link is employed for out-of-band management of the servers and initial cluster deployment over the network from the bastion node. The Intel® Server System S2600WF has a dedicated remote management module (RMM) for the in-band management (IMM). The IMM enables remote-manage capabilities for the servers, access to the server’s remote console for troubleshooting, and running the IPMI commands via the embedded baseboard management controller (BMC) module.
<table>
<thead>
<tr>
<th>Device Name</th>
<th>Device Type</th>
<th>Device Role</th>
<th>IMM Switch Port</th>
<th>25GB #1 Switch Port</th>
<th>25GB #1 Switch Port</th>
<th>Node Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>1GbE Switch</td>
<td>L2 Management switch</td>
<td>Management switch</td>
<td>15</td>
<td>Ethernet4/3</td>
<td>Ethernet4/3</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>100GbE switch 2</td>
<td>Arista DCS-7060CX-32S</td>
<td>Data switch</td>
<td>14</td>
<td>Ethernet4/2</td>
<td>Ethernet4/2</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>100GbE switch 1</td>
<td>Arista DCS-7060CX-32S</td>
<td>Data switch</td>
<td>13</td>
<td>Ethernet4/1</td>
<td>Ethernet4/1</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node15</td>
<td>1u Intel® Server System S2600WF</td>
<td>Bastion node</td>
<td>12</td>
<td>Ethernet3/4</td>
<td>Ethernet3/4</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node14</td>
<td>1u Intel® Server System S2600WF</td>
<td>Master node</td>
<td>11</td>
<td>Ethernet3/3</td>
<td>Ethernet3/3</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node13</td>
<td>1u Intel® Server System S2600WF</td>
<td>Master node</td>
<td>10</td>
<td>Ethernet3/2</td>
<td>Ethernet3/2</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node12</td>
<td>1u Intel® Server System S2600WF</td>
<td>Master node</td>
<td>9</td>
<td>Ethernet3/1</td>
<td>Ethernet3/1</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node11</td>
<td>1u Intel® Server System S2600WF</td>
<td>Infra node</td>
<td>8</td>
<td>Ethernet2/4</td>
<td>Ethernet2/4</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node10</td>
<td>1u Intel® Server System S2600WF</td>
<td>Infra node</td>
<td>7</td>
<td>Ethernet2/3</td>
<td>Ethernet2/3</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node09</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>6</td>
<td>Ethernet2/2</td>
<td>Ethernet2/2</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node08</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>5</td>
<td>Ethernet2/1</td>
<td>Ethernet2/1</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node07</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>4</td>
<td>Ethernet1/4</td>
<td>Ethernet1/4</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node06</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>3</td>
<td>Ethernet1/3</td>
<td>Ethernet1/3</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node05</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>2</td>
<td>Ethernet1/2</td>
<td>Ethernet1/2</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node04</td>
<td>1u Intel® Server System S2600WF</td>
<td>App node</td>
<td>1</td>
<td>Ethernet1/1</td>
<td>Ethernet1/1</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node03</td>
<td>2u Intel® Server System S2600WF</td>
<td>Storage node</td>
<td>3</td>
<td>Ethernet1/3</td>
<td>Ethernet1/3</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node02</td>
<td>2u Intel® Server System S2600WF</td>
<td>Storage node</td>
<td>2</td>
<td>Ethernet1/2</td>
<td>Ethernet1/2</td>
<td>enp24s0f0</td>
</tr>
<tr>
<td>node01</td>
<td>2u Intel® Server System S2600WF</td>
<td>Storage node</td>
<td>1</td>
<td>Ethernet1/1</td>
<td>Ethernet1/1</td>
<td>enp24s0f0</td>
</tr>
</tbody>
</table>
Provisioning

Bastion Node Configuration

The Red Hat OpenShift Container Platform bastion node should be installed with Red Hat Enterprise Linux version 7.4 or newer. The user can choose their preferred installation method. Configure the bastion node with both public and private networks. For IMPI execution, provide access to the management network. All other nodes should be accessible via this native vlan through a PXE interface. The bastion node should be registered with a Red Hat OpenShift subscription. At this step, users should use their individual access to the Red Hat Customer Portal in order to register the bastion node.

```
# subscription-manager register
<username>
<password>
# subscription-manager attach --pool=<openshift_pool_id>
```

Next, enable proper repositories and install software required for bare-metal node provisioning. This will use Ansible Automation, Docker-engine, and Ipmitool.

```
# subscription-manager repos --enable=rhel-7-server-rpms
# subscription-manager repos --enable=rhel-7-server-extras-rpms
# yum install -y ansible docker ipmitool
```

Next, perform an automatic deployment of provisioning engine based on iPXE (an open-source implementation of Preboot eXecution Environment). This is a very effective way of provisioning Red Hat Enterprise Linux Atomic Host operating system to bare-metal servers. Feel free to use an alternative OS provisioning method if you prefer.

To deploy provisioning engine based on iPXE, clone the github repository with Ansible playbooks or download the archive.

```
$ git clone --branch v2.0 https://github.com/intel/openshift-container-architecture.git
```

```
$ wget https://github.com/intel/openshift-container-architecture/archive/v2.0.tar.gz
$ tar -zxvf v2.0.tar.gz
```

Next, set parameters in Ansible inventory file 3. Three parameters need to be set. `bastion_ip` is an IP address of the bastion server. The DHCP range for provisioned servers are set with `dhcp_first_ip` and `dhcp_last_ip`.

```
bastion_ip=172.30.4.10
dhcp_first_ip=172.30.4.100
dhcp_last_ip=172.30.4.150
```

Next, proper iPXE configuration files and kickstarts should be generated for operating systems. Ansible Automation inventory file `/etc/ansible/hosts` must be properly configured. The full inventory used in this reference architecture may be found in Appendix A.

Configuration files are generated based on the `nodes` and `OSEv3:vars` sections. Every server must be placed in `nodes` section and be configured with the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>Serial number</td>
<td>Serial number of server that will be provisioned via iPXE</td>
</tr>
<tr>
<td>ipmi</td>
<td>IP address</td>
<td>IPMI address of server</td>
</tr>
<tr>
<td>openshift_ip</td>
<td>IP address</td>
<td>Private IP address of server [optional]</td>
</tr>
<tr>
<td>openshift_public_ip</td>
<td>IP address</td>
<td>Public IP address of server [optional]</td>
</tr>
<tr>
<td>openshift_hostname</td>
<td>Fully qualified domain name (FQDN)</td>
<td>Hostname of server</td>
</tr>
</tbody>
</table>

Below is the `nodes` section.

```
[nodes]
master1.ocp.example.local openshift_ip=172.30.4.11
openshift_hostname=master1.ocp.example.local
ipmi=192.168.25.12 serial=BQF973900001
[...]
infra1.ocp.example.local openshift_ip=172.30.4.14 openshift_public_ip=100.65.0.14 openshift_hostname=infra1.ocp.example.local
openstack_ip=192.168.25.15 serial=BQF973900006
[...]
```
Besides node specific parameters, there are also few global parameters that should be set in **OSEv3:vars** section. They are listed in **Table 9** below.

### Table 9. iPXE OSEv3:vars parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>external_interface</td>
<td>NIC name</td>
<td>Interface name on which public network is served [optional if external network is not used]</td>
</tr>
<tr>
<td>external_vlan</td>
<td>Vlan id</td>
<td>Vlan id that should be configured on external interface [optional if external network is not used or serves on native vlan]</td>
</tr>
<tr>
<td>external_gateway</td>
<td>IP</td>
<td>IP address of gateway in public network [optional if public network is not used]</td>
</tr>
<tr>
<td>external_netmask</td>
<td>IP netmask</td>
<td>Netmask of public network [optional if public network is not used]</td>
</tr>
<tr>
<td>external_dns</td>
<td>IP</td>
<td>IP address of DNS server in public network [optional if public network is not used or there is no DNS server]</td>
</tr>
<tr>
<td>internal_interface</td>
<td>NIC name</td>
<td>Interface name on which private network is served [optional if private network is not used]</td>
</tr>
<tr>
<td>internal_vlan</td>
<td>Vlan id</td>
<td>Vlan id that should be configured on internal interface [optional if private network is not used or serves on native vlan]</td>
</tr>
<tr>
<td>internal_gateway</td>
<td>IP</td>
<td>IP address of gateway in private network [optional if public network is not used]</td>
</tr>
<tr>
<td>internal_netmask</td>
<td>IP netmask</td>
<td>Netmask of private network [optional if private network is not used]</td>
</tr>
<tr>
<td>local_dns</td>
<td>IP</td>
<td>IP address of DNS server in private network [optional if private network is not used or there is no DNS server]</td>
</tr>
<tr>
<td>root_password</td>
<td>Root account password</td>
<td>Root password to be set on the provisioned systems</td>
</tr>
</tbody>
</table>

Below is the example of **OSEv3:vars** section. It assumes that external connectivity is possible through VLAN 2017. Only servers that have the `openshift_public_ip` parameter configured will be assigned a public network. As internal connectivity is configured on the native VLAN, the parameter `internal_vlan` is not specified.

```
[OSEv3:vars]
external_interface=bond0
external_netmask=255.255.252.0
external_vlan=2017
external_gateway=100.65.0.1
internal_interface=bond0
internal_netmask=255.255.255.0
internal_gateway= {{ bastion_ip }}
local_dns=172.30.4.10
root_password=NODES_ROOT_PASSWORD
```

Additionally, firewall and NAT is configured on the bastion node. If the internal gateway is set to the IP of bastion node then all Red Hat OpenShift nodes will have external communication through its NAT.

**Bare-Metal Provisioning**

This reference architecture uses Red Hat Enterprise Linux Atomic Host. Obtain the ISO image from the Red Hat Customer Portal and transfer it to the bastion node. To use the automatic provisioning method, image content must be copied to proper directories.

```
$ mount -o loop rhel-atomic-installer-7.4.0-1.x86_64.iso /mnt
$ cp -r /mnt/* /tftp/atomic/media/
$ umount /dev/loop0
```

When all environment parameters are set, users can start the automation script to populate configuration files.

The automation script deploys two docker containers. They can be controlled via systemd units. The **Nginx** container serves operating system images via HTTP for maximum performance. The **Dnsmasq** container controls the provisioning process with PXE.

Provisioning system delivers OS image, configuration templates, and network initialization.

Note, that **ipxe.efi** file is downloaded from external sources.

The following script performs the bare-metal provisioning.

```
$ export IPMI_PASSWORD=PASSWORD
$ /tftp/reboot.sh -b pxe -r -f /tftp/ipmi.list.txt
```
This script restarts bare-metal servers and requests a one-time PXE boot. Note that all servers should have the hard drive set as the primary boot device. It uses three flags: b stands for next-boot method (pxe), r stands for reboot request, and f stands for the file with the list of servers with IPMI interfaces that are to be provisioned.

It should just take a few minutes for the servers to be ready. They can be accessed with common credentials. The servers are ready for the next step of OpenShift Container Platform deployment.

Network Configuration

Thanks to Arista Extensible Operating System (EOS*), the Arista DCS-7060CX-32S has an open and programmable network infrastructure. This reference architecture leverages these management capabilities to implement automated network provisioning.

Automatic Switch Provisioning

This reference architecture contains a playbook capable of configuring Arista switch for this OpenShift deployment. Automated tasks in the playbook include enabling eAPI with some security best practices, setting up VLANs, port aggregation, and port configuration.

The following switches specify the main inventory file with switch information:

```
[arista]
Hostname-1 ansible_host=IP_ADDRESS ansible_user=SWITCH_ADMIN_USER ansible_password=SWITCH_ADMIN_PASSWORD
Hostname-2 ansible_host=IP_ADDRESS ansible_user=SWITCH_ADMIN_USER ansible_password=SWITCH_ADMIN_PASSWORD
```

The following enables external access using CLI and creates user with administrative privileges:

```
Arista1#conf t
Arista1(config)#user SWITCH_ADMIN_USER privilege 15 secret 0 SWITCH_ADMIN_PASSWORD
```

Enter the directory containing switch configuration component and run playbook with the appropriate python path:

```
# ansible-playbook openshift-container-architecture/src/eos-configuration/configure-eos.yaml
```

After completing the playbook, network interfaces on the switches are configured according to the reference architecture.

### Prerequisites

#### Inventory

In order to perform initial configuration and installation of the Red Hat OpenShift Container Platform cluster, an Ansible Automation inventory file has to be created with the environment’s description. A full inventory for this reference architecture can be found in Appendix A. Copy this inventory to the `/etc/ansible/hosts` file. All sections of the inventory file specific to Red Hat OpenShift for this reference architecture are described in this chapter. The following chapters provide additional variables that are used for automatic prerequisites and Keepalived deployment.

```
[OSEv3:children]
  nodes
  masters
  glusterfs
  glusterfs_registry
  etcd
  lb
  local
```

This section specifies the types of nodes that are used in an OpenShift Container Platform environment. Required groups are `nodes`, `masters`, and `etcd`. Optional groups are `glusterfs` (persistent storage for containers), `lb` (for load balancing in multi-master clusters), and `local` (which specifies the bastion node).

```
[OSEv3:vars]
  ansible_ssh_user=openshift
  ansible_become=true
  openshift_master_cluster_method=native
  openshift_master_cluster_hostname=ocp.example.local
  openshift_master_cluster_public_hostname=ocp.example.com
  deployment_type=openshift-enterprise
  openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind': 'HTTPPasswordIdentityProvider', 'filename': '/etc/origin/master/users.htpasswd'}]
  openshift_master_htpasswd_users=('[admin': '$apr1$vC6GcVUP$AHZDUSB6AFF6dDMkzF9G1')
  os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
  openshift_hosted_registry_storage_kind=glusterfs
  openshift_hosted_registry_storage_volume_size=200Gi
  openshift_storage_glusterfs_registry_storageclass=True
```
This section describes global cluster parameters.

- **Parameter** `openshift_master_cluster_method` specifies the load balancing method in a multi-master environment. The native value specifies a separated HAProxy load balancer installed on the specified host and configured for the whole environment.

- The hostname for users and cluster components to access the cluster load balancer from external and internal networks is set in the `openshift_master_cluster_hostname` and `openshift_master_cluster_public_hostname` parameters.

- The parameter `openshift_master_identity_providers` configures the authentication method of OpenShift users. In this example, this parameter is based on htpasswd files stored in the Red Hat OpenShift configuration directory. However, you can use many other authentication methods like LDAP, OpenStack Keystone*, OpenID, Gitlab, or GitHub* accounts.

- **`openshift_master_ftpsswd_users`** provides a list of users that can authenticate in OpenShift Container Platform via htpasswd authentication method. In this example there is one user `admin` specified with authentication password `openshift` which is provided in htpasswd hash.

- **`os_sdn_network_plugin_name`** specifies the SDN Open vSwitch plugin used in this environment. In this RA, `redhat/openshift-ovs-multitenant` provides isolation between OpenShift projects on the network level.

- Finally, the last two sections, `openshift_hosted_registry_storage_kind` and `openshift_hosted_registry_storage_volume_size`, specify the storage backend type and its size for Docker Registry. In this solution, Docker Registry uses GlusterFS backed volume for Docker image storage.

This section describes which servers act as Red Hat OpenShift masters. In this reference architecture, three Red Hat OpenShift masters are implemented for control plane HA purposes. Red Hat OpenShift master components can be installed with two methods: rpm-based or container-based. In this RA, all Red Hat OpenShift components are implemented as containers, which is determined by the `containerized=True` parameter.

```bash
[masters]
master1.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.11 openshift_hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.12 openshift_hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.13 openshift_hostname=master3.ocp.example.local
```

```bash
[nodes]
master1.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.11 openshift_hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.12 openshift_hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.13 openshift_hostname=master3.ocp.example.local
infra1.ocp.example.local containerized=True openshift_schedulable=True openshift_node_labels="{'region': 'infra'}" openshift_ip=172.30.4.14 openshift_hostname=infra1.ocp.example.local
infra2.ocp.example.local containerized=True openshift_schedulable=True openshift_node_labels="{'region': 'infra'}" openshift_ip=172.30.4.15 openshift_hostname=infra2.ocp.example.local
app1.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.16 openshift_hostname=app1.ocp.example.local
app2.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.17 openshift_hostname=app2.ocp.example.local
app3.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.18 openshift_hostname=app3.ocp.example.local
app4.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.19 openshift_hostname=app4.ocp.example.local
app5.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.20 openshift_hostname=app5.ocp.example.local
app6.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.21 openshift_hostname=app6.ocp.example.local
gluster1.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.22 openshift_hostname=gluster1.ocp.example.local
gluster2.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.23 openshift_hostname=gluster2.ocp.example.local
gluster3.ocp.example.local containerized=True openshift_schedulable=True openshift_ip=172.30.4.24 openshift_hostname=gluster3.ocp.example.local
```
This section describes which servers act as OpenShift nodes. In this reference architecture, seven OpenShift nodes are implemented. Two of them perform infrastructure functions, which is determined by the `openshift_node_labels="{'region': 'infra'}"` parameter. OpenShift node components are also installed on OpenShift master and GlusterFS storage servers. However, no user application should be deployed on OpenShift masters because of the `openshift_schedulable=False` parameter. All other nodes have scheduling enabled. In this reference architecture, all node components are implemented as containers, which is determined by the `containerized=True` parameter.

This section describes which servers act as GlusterFS nodes. In this reference architecture, three containerized GlusterFS nodes are implemented on dedicated hosts. Additionally, the file specifies which of their IP addresses and disks act as GlusterFS volumes. This cluster is used as a default storage backend based on SSD drives. It also provides a persistent volume for the OpenShift private Docker registry.

This section describes which servers act as additional GlusterFS nodes for storage backend based on NVMe drives. This provides an additional level of storage available for Red Hat OpenShift users and can be used when specific storage for intense workloads is required.

This section describes hosts that will run etcd instances. In this reference architecture, three etcd instances are installed on three master servers to achieve low-latency traffic between them. When many etcd instances are specified in an inventory file, they are automatically clustered in order to provide a highly available key-value etcd store. An etcd cluster that consists of three etcd instances resists a failure of one etcd instance. It is recommended to have an odd number of etcd instances in a cluster.

When `openshift_master_cluster_method` is set to `native`, this section specifies a host on which HAProxy load balancer will be installed and configured. In this reference architecture, two HAProxy load balancers are installed on two infrastructure servers. They use one common virtual IP address that is managed by Keepalived to provide a highly available OpenShift Container Platform cluster.

### Node Preparation

When the operating system deployment is finished, the nodes must be prepared for the Red Hat OpenShift installation.

Preliminary steps are described in the Red Hat OpenShift documentation (https://docs.openshift.com/container-platform/3.6/). Note that this reference architecture provides Ansible Playbooks for automating prerequisites. Those playbooks can be downloaded from the github repository https://github.com/intel/openshift-container-architecture.

1. Prepare an openshift account and exchange SSH keys across all nodes.
2. Attach software subscriptions.
3. Install and configure the DNS service.
4. Install additional packages.
5. Configure Docker Engine.
Automatic Prerequisites Installation

All required tasks are prepared as Ansible Playbooks, which are ready to use and available at https://github.com/intel/openshift-container-architecture. You can use those playbooks to prepare all needed tasks automatically instead of completing the manual steps.

In the OSEv3:vars section in ansible inventory file, set up additional variables as shown in Table 10.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhel_subscription_user</td>
<td>Name of the user who will be used for registration</td>
</tr>
<tr>
<td>rhel_subscription_pass</td>
<td>Password of the user who will be used for registration</td>
</tr>
<tr>
<td>ansible_ssh_user</td>
<td>Insert root or other user with root privileges</td>
</tr>
<tr>
<td>ansible_become</td>
<td>Set to True to run commands with sudo privileges</td>
</tr>
<tr>
<td>local_dns</td>
<td>Type a proper IP address for your bastion node that runs the DNS service</td>
</tr>
</tbody>
</table>

To deploy Keepalived daemons using an Ansible Playbook on infra nodes, enter following command inside the forked Git repository:

```
$ su openshift
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/adhoc/uninstall.yml
```

### Red Hat OpenShift Container Platform Deployment

#### Red Hat OpenShift Container Platform Installation

When the inventory file with the environment description is prepared and all prerequisites are configured, you can perform OpenShift Container Platform installation from the bastion host. This process is simple and requires a single command:

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/byo/config.yml
```

After the installation process, the Ansible Playbook should report no errors. The OpenShift Container Platform environment will be set up. Note that if needed, you can easily uninstall the environment with the following command:

```
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/adhoc/uninstall.yml
```

### Table 10. Additional inventory variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhel_subscription_user</td>
<td>Name of the user who will be used for registration</td>
</tr>
<tr>
<td>rhel_subscription_pass</td>
<td>Password of the user who will be used for registration</td>
</tr>
<tr>
<td>ansible_ssh_user</td>
<td>Insert root or other user with root privileges</td>
</tr>
<tr>
<td>ansible_become</td>
<td>Set to True to run commands with sudo privileges</td>
</tr>
<tr>
<td>local_dns</td>
<td>Type a proper IP address for your bastion node that runs the DNS service</td>
</tr>
</tbody>
</table>

In order to use automatic Keepalived Ansible Playbooks, the following variables must be defined in the OSEv3:vars section in the Ansible Automation inventory:

```
external_interface=bond0
external_vlan=2017
internal_interface=bond0
openshift_master_cluster_ip=172.30.4.30
openshift_master_cluster_public_ip=100.65.0.30
```

While Keepalived can be deployed manually, this RA includes Ansible Playbooks for automatic Keepalived deployment.

### Red Hat OpenShift Container Platform Deployment

#### Automatic Keepalived Deployment

OpenShift Container Platform delivers two flavors of HAProxy load balancing software. The first flavor, spawned as a Docker container, distributes API calls between master servers. The second flavor, spawned as a Docker container, provides the router mechanism for exposing applications inside a cluster. For HA, maximum fault tolerance, and performance, this reference architecture includes an additional Keepalived component. Keepalived is open-source software distributed under GPL license and is used to provide simple and robust load balancing and HA for the HAProxy instances. It is recognized by Red Hat as a recommended solution, and this implementation is based on the official Red Hat Enterprise Linux documentation.

This reference architecture uses HAProxy instances in both flavors. They are spawned as Docker containers on both infra nodes. In conjunction with the floating IP address provided by Keepalived, this arrangement mitigates a single point of failure. Installation and configuration can be performed manually or through a single Ansible Playbook command.

While Keepalived can be deployed manually, this RA includes Ansible Playbooks for automatic Keepalived deployment.
Deployment Validation

When installation completes without any errors, you should also validate the deployment. There are a few components you should check. First, log on to one of the Red Hat OpenShift master nodes and check if all nodes are connected to the cluster:

```
$ ssh master1.ocp.example.local
$ oc get nodes
NAME       STATUS             AGE    
master1.ocp.example.local Ready,SchedulingDisabled 1h
master2.ocp.example.local Ready,SchedulingDisabled 1h
master3.ocp.example.loc Ready,SchedulingDisabled 1h
infra1.ocp.example.loc Ready 1h
infra2.ocp.example.loc Ready 1h
app1.ocp.example.loc Ready 1h
app2.ocp.example.loc Ready 1h
app3.ocp.example.loc Ready 1h
app4.ocp.example.loc Ready 1h
app5.ocp.example.loc Ready 1h
app6.ocp.example.loc Ready 1h
gluster1.ocp.example.loc Ready 1h
gluster2.ocp.example.loc Ready 1h
gluster3.ocp.example.loc Ready 1h
```

You can use the above command to verify OpenShift node states. All cluster nodes should be listed and marked as **Ready**. If any node is in a **NotReady** state then it is not properly assigned to a cluster and should be inspected.

You can use the following command to verify the etcd cluster state. All etcd members should be listed and marked as **healthy**. If any etcd member is in an **unhealthy** state then it is not properly assigned to an etcd cluster and should be further inspected.

```
$ sudo etcdctl -C https://etcd1.ocp.example.local:2379 --ca-file=/etc/etcd/ca.crt --cert-file=/etc/etcd/peer.crt --key-file=/etc/etcd/peer.key cluster-health
```

```
member 5f0aab880290ddeb is healthy: got healthy result from https://etcd1.ocp.example.local:2379
member c305190f3c57613c is healthy: got healthy result from https://etcd2.ocp.example.local:2379
member c434590bbf158f3d is healthy: got healthy result from https://etcd3.ocp.example.local:2379
```

Log on to the OpenShift Container Platform web console using the following URL address: [https://ocp.example.com:8443](https://ocp.example.com:8443) to verify that you can create a project, services, and other Red Hat OpenShift application components.

To verify the infrastructure node components of the OpenShift Container Platform cluster, type the following command. It should return a list of pods that run Docker Registry and router services. They all should have a **Running** status.

```
$ oc get pods --namespace=default
NAME                      READY STATUS RESTARTS AGE
docker-registry-2-qql92   1/1 Running 0 1h
docker-registry-2-uh7op   1/1 Running 0 1h
router-1-2vgcm            1/1 Running 0 1h
router-1-cbz87            1/1 Running 0 1h
```

Type the following command to verify the GlusterFS storage backend components of the OpenShift Container Platform cluster. It returns a list of pods that run GlusterFS and Heketi services. They all should have a status of **Running**. Three GlusterFS nodes serve as a default SSD storage cluster and six GlusterFS nodes serve as an additional NVMe storage cluster.

```
$ oc get pods --namespace=glusterfs
NAME                  READY STATUS RESTARTS AGE
glusterfs-nvme-5r0dl   1/1 Running 0 1h
glusterfs-nvme-bjbfs   1/1 Running 0 1h
glusterfs-nvme-gcp2r   1/1 Running 0 1h
glusterfs-nvme-9xtrj   1/1 Running 0 1h
glusterfs-nvme-si26z   1/1 Running 0 1h
glusterfs-nvme-tc3b1   1/1 Running 0 1h
glusterfs-ssd-jqj7l    1/1 Running 0 1h
glusterfs-ssd-lspb4    1/1 Running 0 1h
glusterfs-ssd-tchn4    1/1 Running 0 1h
heketi-nvme-1-3x9fs    1/1 Running 0 1h
heketi-ssd-1-5zxzf     1/1 Running 0 1h
```
Summary and Conclusions

Solutions involving Red Hat OpenShift Container Platform and Red Hat container-native storage are created to deliver a production-ready foundation that simplifies the deployment process, shares the latest best practices, and provides a stable, highly available environment on which to run your production applications. This reference architecture covered the process of provisioning and deploying a highly available OpenShift Container Platform cluster on a private cloud environment with both the registry and the application pods backed by Red Hat container-native storage.

For any questions or concerns, please contact your account representative or visit Github to provide input on the product: https://github.com/intel/openshift-containerarchitecture/issues.

Authors
Łukasz Łuczaj
Jose Palafox
Łukasz Sztachański
David Cain
[OSEv3:children]
masters
nodes
etcd
lb
local
glusterfs
glusterfs_registry

[OSEv3:vars]
ansible_ssh_user=openshift
ansible_become=true
openshift_master_cluster_method=native
openshift_master_cluster_hostname=ocp.example.local
openshift_master_cluster_public_hostname=ocp.example.com
openshift_master_default_subdomain=apps.ocp.example.com
openshift_master_cluster_ip=172.30.4.30
openshift_master_cluster_public_ip=100.65.0.30
openshift_master_portal_net=10.0.0.0/16
deployment_type=openshift-enterprise
openshift_release=v3.6
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/users.htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$vC6GcVU$AHZDU5BAFF6dDMfk.IFzG1'}
rhel_subscription_user=user@example.com
rhel_subscription_pass=portal_password
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=200Gi
openshift_storage_glusterfs_registry_storageclass=True
local_dns=172.30.4.10
external_interface=bond0

external_vlan=2017
external_netmask=255.255.250.0
external_gateway=100.65.0.1
internal_interface=bond0
internal_netmask=255.255.255.0
bastion_ip=172.30.4.10
internal_gateway=[‘(bastion_ip)]
dhcp_first_ip=172.30.4.100
dhcp_last_ip=172.30.4.150
root_password=NODE_ROOT_PASSWORD

[local]
127.0.0.1

[masters]
master1.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.11 openshift_hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.12 openshift_hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True openshift_schedulable=False openshift_ip=172.30.4.13 openshift_hostname=master3.ocp.example.local

[nodes]
master1.ocp.example.local openshift_schedulable=False openshift_ip=172.30.4.11 openshift_hostname=master1.ocp.example.local ipmi=192.168.25.12 serial=BQF973900001
master2.ocp.example.local openshift_schedulable=False openshift_ip=172.30.4.12 openshift_hostname=master2.ocp.example.local ipmi=192.168.25.13 serial=BQF973900002
master3.ocp.example.local openshift_schedulable=False openshift_ip=172.30.4.13 openshift_hostname=master3.ocp.example.local ipmi=192.168.25.14 serial=BQF973900003
infra1.ocp.example.local openshift_node_labels="{\"region\": \"infra\"}" openshift_schedulable=True containerized=True openshift_public_ip=100.65.0.14 openshift_hostname=infra1.ocp.example.local ipmi=192.168.25.15 serial=BQF973900006
infra2.ocp.example.local openshift_node_labels="{'region':'infra'}", openshift_schedulable=True, containerized=True, openshift_public_ip=100.65.0.15, openshift_ip=172.30.4.15, openshift_hostname=infra2.ocp.example.local, ipmi=192.168.25.16, serial=BQF973900004

app1.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.16, openshift_hostname=app1.ocp.example.local, ipmi=192.168.25.17, serial=BQF973900007

app2.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.17, openshift_hostname=app2.ocp.example.local, ipmi=192.168.25.18, serial=BQF973900008

app3.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.18, openshift_hostname=app3.ocp.example.local, ipmi=192.168.25.19, serial=BQF973900009

app4.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.19, openshift_hostname=app4.ocp.example.local, ipmi=192.168.25.20, serial=BQF973900010

app5.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.20, openshift_hostname=app5.ocp.example.local, ipmi=192.168.25.21, serial=BQF973900011

app6.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.21, openshift_hostname=app6.ocp.example.local, ipmi=192.168.25.22, serial=BQF973900012

app7.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.22, openshift_hostname=app7.ocp.example.local, ipmi=192.168.25.23, serial=BQF9794100020

gluster1.ocp.example.local glusterfs_ip=172.30.4.22, glusterfs_devices="['/dev/nvme0n1']"

app8.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.23, openshift_hostname=app8.ocp.example.local, ipmi=192.168.25.24, serial=BQF9794100021

app9.ocp.example.local containerized=True, openshift_scheduled=True, openshift_ip=172.30.4.24, openshift_hostname=app9.ocp.example.local, ipmi=192.168.25.25, serial=BQF9794100023

[etcd]
etcd1.ocp.example.local containerized=True, openshift_public_ip=172.30.4.11, openshift_hostname=etcd1.ocp.example.local

etcd2.ocp.example.local containerized=True, openshift_public_ip=172.30.4.12, openshift_hostname=etcd2.ocp.example.local

etcd3.ocp.example.local containerized=True, openshift_public_ip=172.30.4.13, openshift_hostname=etcd3.ocp.example.local

[lb]
lb1.ocp.example.local containerized=True, openshift_public_ip=100.65.0.14

lb2.ocp.example.local containerized=True, openshift_public_ip=100.65.0.15

[glusterfs]
appl1.ocp.example.local glusterfs_ip=172.30.4.16, glusterfs_devices="['/dev/nvme0n1']"
appl2.ocp.example.local glusterfs_ip=172.30.4.17, glusterfs_devices="['/dev/nvme0n1']"
appl3.ocp.example.local glusterfs_ip=172.30.4.18, glusterfs_devices="['/dev/nvme0n1']"
appl4.ocp.example.local glusterfs_ip=172.30.4.19, glusterfs_devices="['/dev/nvme0n1']"
appl5.ocp.example.local glusterfs_ip=172.30.4.20, glusterfs_devices="['/dev/nvme0n1']"
appl6.ocp.example.local glusterfs_ip=172.30.4.21, glusterfs_devices="['/dev/nvme0n1']"

gluster1.ocp.example.local glusterfs_ip=172.30.4.22, glusterfs_devices="['/dev/sdb', '/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf', '/dev/sdg', '/dev/sdh', '/dev/sdi', '/dev/sdj', '/dev/sdk', '/dev/sdl', '/dev/sdm', '/dev/sdn', '/dev/sdo', '/dev/bsd', '/dev/sdq', '/dev/sdr', '/dev/sds', '/dev/sdt', '/dev/sdu']"
gluster2.ocp.example.local glusterfs_ip=172.30.4.23, glusterfs_devices="['/dev/sdb', '/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf', '/dev/sdg', '/dev/sdh', '/dev/sdi', '/dev/sdj', '/dev/sdk', '/dev/sdl', '/dev/sdm', '/dev/sdn', '/dev/sdo', '/dev/bsd', '/dev/sdq', '/dev/sdr', '/dev/sds', '/dev/sdt', '/dev/sdu']"
'/dev/sdg', '/dev/sdh', '/dev/sdi', '/dev/sdj', '/dev/sdk', '/dev/sdk', '/dev/sdm', '/dev/sdn', '/dev/sdo', '/dev/sdp', '/dev/sdq', '/dev/sdr', '/dev/sds', '/dev/sdt', '/dev/sdu']

```python
'gluster3.ocp.example.local glusterfs_ip=172.30.4.24 glusterfs__
devices="['/dev/sdb', '/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf',
'/dev/sdg', '/dev/sdh', '/dev/sdi', '/dev/sdj', '/dev/sdk', '/dev/sdk',
'/dev/sdm', '/dev/sdn', '/dev/sdo', '/dev/sdp', '/dev/sdq', '/dev/sdr',
'/dev/sds', '/dev/sdt', '/dev/sdu']"

[arista]
arista-1 ansible_host=192.168.25.2 ansible_user=admin
ansible_password=SWITCH_ADMIN_PASSWORD
arista-2 ansible_host=192.168.25.3 ansible_user=admin
ansible_password=SWITCH_ADMIN_PASSWORD