The term “cloud computing” is often associated with virtual machines, but many emerging and rapidly growing cloud technologies now make use of containerization. Containers can now be used as an alternative to OS-level virtualization to run multiple isolated applications on a single host with a much smaller footprint than virtual machines require. Container-based virtualization offers many benefits when compared to traditional virtualization technologies, and containers are perceived as an even more portable and faster way to deploy services on cloud infrastructures.

While containers themselves provide many benefits, they are not easily manageable in large environments. That's why many container orchestration tools have increased in momentum and gained popularity. Each orchestration tool is different, however, and should be chosen individually for specific purposes.

This reference architecture (RA) will show you how to prepare, provision, deploy, and manage a Red Hat OpenShift Container Platform 3.5–based private cloud environment. The intended audience for this RA is system administrators or system architects. Some experience with Docker* and OpenShift technologies might be helpful, but is not required.
Hardware Summary

In this RA, the following hardware is included:

- **Lenovo System x3550 M5 rack servers**, which are 1U, two-socket servers that can handle complex workloads, including big data and virtualization
- **Two (2) Lenovo ThinkSystem NE10032 RackSwitch switches**, which are data plane, entry-level layer-2 switches
- **Lenovo RackSwitch™ G7052 switch**, which is a management plane, entry-level layer-2 switch

Software Summary

In this RA, the following software is included:

- **Extreme Cluster/Cloud Administration Toolkit* (xCAT*)** to deploy the operating system for the Red Hat Enterprise Linux (RHEL) (for bastion and infra) and Red Hat Enterprise Linux Atomic Host (for master and application nodes)
- **RHEL and Red Hat Enterprise Linux Atomic Host** operating systems installed on OpenShift cluster nodes
- **OpenShift Container Platform**, which 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
- **Lenovo™ XClarity® Administrator** for management of the operating systems on bare-metal servers

Furthermore, 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
- **Ansible®** 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
- **A Network File System (NFS) server** for storing application images and providing persistent volumes (PV) functionality
- **Distributed Replicated Block Device* (DRBD*)** to provide a backup device for Docker images distributed via the NFS server

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>7.3</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux Atomic Host</td>
<td>7.3.3</td>
</tr>
<tr>
<td>OpenShift Container Platform</td>
<td>3.5</td>
</tr>
<tr>
<td>Docker</td>
<td>1.12.6</td>
</tr>
<tr>
<td>Ansible</td>
<td>2.2.1.0</td>
</tr>
<tr>
<td>Etcd</td>
<td>3.1.0</td>
</tr>
<tr>
<td>Open vSwitch</td>
<td>2.6.1</td>
</tr>
<tr>
<td>HAProxy</td>
<td>1.5.18</td>
</tr>
<tr>
<td>Keepalived</td>
<td>1.2.13</td>
</tr>
<tr>
<td>DRBD</td>
<td>8.9.8</td>
</tr>
</tbody>
</table>
Red Hat Enterprise Linux Atomic Host

This RA uses two types of operating systems—Red Hat Enterprise Linux 7.3 (for bastion and infrastructure nodes) and Red Hat Enterprise Linux Atomic Host 7.3.3 (for master and application nodes). RHEL Atomic Host is a lightweight variant of the RHEL operating system, designed to run Linux containers. The RHEL operating system helps ensure high quality and a reliable base for the whole system, provides strong security features, and supports business-critical workloads, interoperability between a variety of operating systems, and much more.

Ansible®

Ansible is an IT automation tool capable of provisioning, deploying applications, and configuration and management of operating system's components and other devices. The OpenShift RA is based largely on Ansible playbooks due to its simplicity and extensibility. That is also the reason why this RA adopts and distributes improvements in the same manner.

OPENSHIFT SYSTEM ARCHITECTURE

The OpenShift Container Platform is a complete container application platform that provides all aspects of the application development process in one consistent solution across multiple infrastructure footprints. 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 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 OpenShift platform is based on the Kubernetes cluster architecture.

This OpenShift RA contains four types of nodes: bastion, master, infrastructure, and application, which are described below.

Bastion Node

This is a dedicated node that serves as the main deployment and management server for the OpenShift cluster. This 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 RHEL 7.3 with the Linux KVM packages installed.

OpenShift Master

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 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 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 RHEL Atomic Host.

OpenShift Infrastructure Node

The OpenShift infrastructure node runs infrastructure-specific services: Docker Registry* and the HAProxy router. Docker Registry stores application images in the form of containers. The HAProxy router provides routing functions for OpenShift applications. It currently supports HTTP(S) traffic and TLS-enabled traffic via Server Name Indication (SNI). Additional applications and services can be deployed on OpenShift infrastructure nodes. The OpenShift infrastructure node runs RHEL 7.3.

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. An OpenShift application node runs RHEL Atomic Host.

Figure 1. OpenShift Node Roles

Hardware Detail

The OpenShift RA is validated using Lenovo servers and network switches. The configuration includes six application nodes, three master nodes, two infrastructure nodes, and one bastion node. Figure 2 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.
Lenovo System x3550 M5 Rack Servers

The Lenovo System x3550 M5 rack server is a 1U, two-socket rack server available in many configurations optimized to handle various workload requirements for compute, storage, and input/output (I/O). The server supports the Intel® Xeon® processor E5-2600 v4 product family and fast, energy-efficient TruDDR4* memory to deliver exceptional performance. Flexible and scalable internal storage configurations include up to twelve 2.5-inch or four 3.5-inch drives with a wide selection of drive sizes and types. Lenovo System x3550 M5 rack servers offer a 12 gigabit per second (Gbps) Serial Attached SCSI (SAS) RAID controller with support for hardware RAID 0/1/10. Flexible and scalable internal storage configurations provide up to 92 TB of storage capacity with 7.68 TB 2.5-inch solid-state drives (SSDs) in a 1U rack form factor. A Lenovo System x3550 M5 rack server has four integrated gigabit Ethernet (GbE) ports and optional 10 GbE ports with mezzanine LOM (ML2) adapters.

Lenovo servers offer industry-leading reliability with Predictive Failure Analysis (PFA) and smart diagnostic tools that help to reduce downtime. Users can also reduce the cost of their infrastructures by using energy-smart features for efficient performance.

The Lenovo x3550 M5 rack server includes an Integrated Management Module II (IMM2.1) to monitor server availability and perform remote management. Provisioning and management of a Lenovo System x3550 M5 rack server is simplified with the Lenovo XClarity enterprise management tool. Lenovo XClarity Administrator offers many enterprise features like automated discovery, inventory tracking, real-time monitoring, and provisioning of the operating system on bare-metal servers.

The Intel® Xeon® Processor E5-2600 v4 Product Family

The Intel Xeon processor E5-2600 v4 product family is designed to help enterprises, cloud service providers, and telecommunications companies accelerate the move toward next-generation software-defined infrastructure (SDI). The Intel Xeon processor E5-2600 v4 product family is manufactured on 14 nm processor technology. It provides more than 20 percent more cores than the Intel Xeon processor E5-2600 v3 product family, and it supports faster memory and includes technologies for accelerating specific workloads. Examples of such technologies include:

- **Intel® QuickPath Interconnect (Intel® QPI)** for fast and resilient system communications with up to 9.6 GT/s of Intel QPI speed per channel
- **Intel® Transactional Synchronization Extensions (Intel® TSX)** for high performance of multi-threaded workloads
The Lenovo™ ThinkSystem® NE10032 RackSwitch

The Lenovo ThinkSystem NE10032 RackSwitch is a newly available 1U top-of-rack (ToR) switch designed to deliver a high performing 100 Gbps data center server-connectivity solution. It offers thirty-two 100 Gbps QSFP28 data ports and one RJ45* 10/100/1000M port for out-of-band management. It is powered by an Intel Atom® processor C2558 with four 2.4 GHz cores, 4 GB RAM, and a 16 GB Serial ATA (SATA) SSD. Redundant power supplies and hot-swappable fans improve reliability in the data center infrastructure. The switch runs the Lenovo Cloud Network Operating System (CNOS) version 10.4.1.x, which provides scalable, open switching with support for scripting and software-defined networking (SDN) designed to scale for business needs. Lenovo ThinkSystem NE10032 RackSwitch management can be easily automated with use of Ansible tools, Python* scripts, and REST APIs.

The Lenovo ThinkSystem NE10032 RackSwitch data ports support configurations of 1x100GbE, 1x40GbE, 2x50GbE, 4x25GbE, or 4x10GbE on the PHY. Depending upon the type of network adapters installed on the server side, such as 100GbE, 40GbE, 25GbE, or 10GbE, the ports’ profiles on the switch need to be configured accordingly. This RA uses Intel® Ethernet Converged Network Adapter X710 DA2 dual-port 10 Gbps SFP+ adapters on the Lenovo System x3550 M5 rack servers. Each of the switch ports is configured in 4x10GbE mode and uses the QSFP28-4xSFP28 breakout cables to connect each switch port to four servers in the cluster.

Figure 4. The Lenovo ThinkSystem NE10032 RackSwitch 100 Gbps switch

Figure 4 shows the Lenovo ThinkSystem NE10032 RackSwitch and the connectivity between the OpenShift cluster nodes and the switch using the QSFP28-4xSFP28 breakout cables.

Hardware Configuration

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

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

Table 2. OpenShift node roles and hardware configuration

<table>
<thead>
<tr>
<th>OpenShift Role</th>
<th>Qty.</th>
<th>Platform</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| Bastion Node      | 1    | Lenovo System x3550 M5 | • 2x Intel® Xeon® processor E5-2630 v4 at 2.20 GHz  
|                   |      |                | • 256 GB memory (16x 16 GB)  
|                   |      |                | • 2x Intel® SSD DC S3520 Series 150 GB enterprise entry SATA G3HS 2.5" SSDs (RAID1)  
|                   |      |                | • 1x ServeRAID® M1215 SAS/SATA controller  
|                   |      |                | • 1x Intel® Ethernet Converged Network Adapter X710, dual-port 10 Gbps SFP+ ML2 |
| Master Node       | 3    | Lenovo System x3550 M5 | • 2x Intel Xeon processor E5-2630 v4 at 2.20 GHz  
|                   |      |                | • 256 GB memory (16x 16 GB)  
|                   |      |                | • 2x Intel SSD DC S3520 Series 150 GB enterprise entry SATA G3HS 2.5" SSD (RAID1)  
|                   |      |                | • 1x ServeRAID M1215 SAS/SATA controller  
|                   |      |                | • 1x Intel Ethernet Converged Network Adapter X710, dual-port 10 Gbps SFP+ ML2 |
| Infrastructure Node | 2  | Lenovo System x3550 M5 | • 2x Intel Xeon processor E5-2630 v4 at 2.20 GHz  
|                   |      |                | • 256 GB memory (16x 16 GB)  
|                   |      |                | • 2x Intel SSD DC S3520 Series 480 GB enterprise entry SATA G3HS 2.5" SSD (RAID1)  
|                   |      |                | • 1x ServeRAID M1215 SAS/SATA controller  
|                   |      |                | • 1x Intel Ethernet Converged Network Adapter X710, dual-port 10 Gbps SFP+ ML2 |
| App node          | 6    | Lenovo System x3550 M5 | • 2x Intel Xeon processor E5-2680 v4 at 2.40 GHz  
|                   |      |                | • 256 GB memory (16x 16 GB)  
|                   |      |                | • 2x Intel SSD DC S3520 Series 480 GB enterprise entry SATA G3HS 2.5" SSD (RAID1)  
|                   |      |                | • 1x ServeRAID M1215 SAS/SATA controller  
|                   |      |                | • 1x Intel Ethernet Converged Network Adapter X710, dual-port 10 Gbps SFP+ ML2 |
As shown in Table 4, there are power options available for 30A single phase, 60A single phase, or 60A three phase power (and the equivalent for non-U.S. countries), depending upon the customer's data center environment. Choosing your power configuration based on the initial cluster configuration and the future scaling requirements can help you avoid expensive reconfiguration of the customer's environment.

In addition, the total power draw requirements for the configured cluster should be calculated to be within the rated power specifications of the PDUs. For example, 30A single phase PDUs could be used for a cluster of four to eight nodes, but 60A single phase or three phase PDUs are needed for clusters of more than eight nodes.

For more information on Lenovo PDUs, see https://support.lenovo.com/us/en/solutions/lvno-powinf.

### Power Configuration

For the OpenShift solution, various power configuration choices are available from Lenovo. We recommend using the Lenovo switched and monitored PDUs, which are optimized for rack integration. In addition, the Lenovo 42U dynamic 1,100 mm rack cabinet provides six vertical side pockets for installing the PDUs without occupying the rack’s horizontal slots, leaving space for installing servers and switches in those slots. Table 4 also provides the list of the standard part numbers for the rack cabinet and PDUs with various power ratings, depending upon the customer’s data center power needs.

The recommended power connectivity is shown in Figure 5. The diagram shows the power connectivity for the full rack with 12 Lenovo System x3550 M5 rack server–based nodes and three Lenovo switches with redundant power supplies. The recommended PDUs are 60 amp single phase 208 VAC PDUs (which are de-rated to 48 A of current in the United States) to support all devices in the rack. Each power domain contains two PDUs. The first eight nodes are wired to PDU 1, and the second set of four nodes and the three switches are wired to PDU 3. The same connections are made to PDUs 2 and 4 in the second power domain.

As shown in Table 4, there are power options available for 30A single phase, 60A single phase, or 60A three phase power (and the equivalent for non-U.S. countries), depending upon the customer’s data center environment. Choosing your power configuration based on the initial cluster configuration and the future scaling requirements can help you avoid expensive reconfiguration of the customer’s environment.

In addition, the total power draw requirements for the configured cluster should be calculated to be within the rated power specifications of the PDUs. For example, 30A single phase PDUs could be used for a cluster of four to eight nodes, but 60A single phase or three phase PDUs are needed for clusters of more than eight nodes.

For more information on Lenovo PDUs, see https://support.lenovo.com/us/en/solutions/lvno-powinf.

### Table 3. The Lenovo and OpenShift RA hardware bill of materials

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
<th>MTM</th>
<th>FC</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x Rack Enclosure</td>
<td>42U 1,200 mm Deep Dynamic Rack (six sidewall compartments)</td>
<td>9363-4PX</td>
<td>7649</td>
<td><a href="https://lenovopress.com/lp0658-lenovo-rack-cabinet-reference">https://lenovopress.com/lp0658-lenovo-rack-cabinet-reference</a></td>
</tr>
<tr>
<td>1x Management Switch</td>
<td>Lenovo RackSwitch G7052 (1Gbps)</td>
<td>7159-HCT</td>
<td>ATOA</td>
<td><a href="https://lenovopress.com/tips1269-lenovo-rackswitch-g7052">https://lenovopress.com/tips1269-lenovo-rackswitch-g7052</a></td>
</tr>
<tr>
<td>2x Cluster Traffic</td>
<td>Lenovo ThinkSystem NE10032 RackSwitch (100 Gbps)</td>
<td></td>
<td></td>
<td><a href="https://lenovopress.com/lp0609.pdf">https://lenovopress.com/lp0609.pdf</a></td>
</tr>
<tr>
<td>Switches</td>
<td>• 1x Bastion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 2x Infrastructure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 3x Management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 6x Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lenovo System x3550 M5 Server</td>
<td>8869</td>
<td></td>
<td><a href="https://lenovopress.com/lp0067-lenovo-system-x3550-m5-machine-type-8869">https://lenovopress.com/lp0067-lenovo-system-x3550-m5-machine-type-8869</a></td>
</tr>
</tbody>
</table>

| 2-4 Rack power         | distribution units (PDUs)                                                 | Refer to Table 4. |      |                                                                      |

### Table 4. Lenovo power distribution units (PDUs)

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Feature</th>
<th>MFI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9363-RC4</td>
<td>A1RC</td>
<td>90Y3067</td>
<td>42U 1,100 mm Enterprise V2 Dynamic Rack</td>
</tr>
<tr>
<td>46M4005</td>
<td>5895</td>
<td>46M4011</td>
<td>1U 12 C13 Switched and Monitored 60A 3 Phase PDU</td>
</tr>
<tr>
<td>46M4004+40K9614</td>
<td>5908</td>
<td>46M4010</td>
<td>(unit), 41Y9256 (line cord) C13 PDU &amp; 1p, 30A/208V, NEMA L6-30P(US)LC</td>
</tr>
<tr>
<td>46M4004+40K9615</td>
<td>5909</td>
<td>46M4010</td>
<td>(unit), 41Y9257 (line cord) C13 PDU &amp; 1p, 60A/208V, IEC 309 2P+G(US)LC</td>
</tr>
<tr>
<td>46M4004+40K9612</td>
<td>5910</td>
<td>46M4010</td>
<td>(unit), 41Y9258 (line cord) C13 PDU &amp;1p, 32A/230V, IEC309 P+N+G(non-US)LC</td>
</tr>
<tr>
<td>46M4004+40K9613</td>
<td>5911</td>
<td>46M4010</td>
<td>(unit), 41Y9259 (line cord) C13 PDU &amp; 1p, 63A/230V, IEC309 P+N+G(non-US)LC</td>
</tr>
</tbody>
</table>

For more information on Lenovo PDUs, see https://support.lenovo.com/us/en/solutions/lvno-powinf.

![Figure 5. Rack power diagram](image-url)
Networking Overview

The Lenovo OpenShift platform uses the 10 GbE network as the primary fabric for inter-node communication. As described previously, two Lenovo ThinkSystem NE10032 RackSwitch switches are part of the solution to provide data layer communication, and one management Lenovo RackSwitch G7052 is used for “out-of-band” communication. Figure 6 shows the network architecture for this solution.

![Figure 6. Virtual Link Aggregation (VLAG)](image)

In this RA, the solution is designed to deliver maximum availability. Virtual Link Aggregation Group (VLAG) is a feature of the Lenovo CNOS operating system that allows a pair of Lenovo switches to work as a single virtual switch. Each of the cluster nodes has a link to each VLAG peer switch for redundancy. This provides improved high availability (HA) for the nodes using the link aggregation control protocol (LACP) for aggregated bandwidth capacity. Connection to the uplink core network is facilitated by the VLAG 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 VLAG peers is an inter-switch link (ISL) and provides excellent support of east-west cluster traffic the nodes. The VLAG 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. (The remaining switch carries all the traffic.)

Network Architecture

There are three logical networks in this RA:

- **External**: The external network is used for the public API, the OpenShift web interface, and exposed applications (services and routes).

- **Internal**: This is the primary, non-routable 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.

- **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 7. OpenShift logical/physical network connectivity](image)

Figure 7 shows the components of the Red Hat OpenShift solution and their logical architecture. All OpenShift nodes are connected via the internal network, where they can communicate with each other. Furthermore, Open vSwitch creates its own network for 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 OpenShift web console and applications. Lastly, there is an NFS server that shares disk space with Docker Registry for Docker image storage. This storage is backed up by Distributed Replicated Block Device (DRBD), so Docker Registry storage can be easily switched in case of a node failure.

![Figure 8. OpenShift Container Platform logical environment](image)
Network Addresses
Table 5 shows the network and subnet addresses for the various networks in the OpenShift cluster.

Table 5. Network addresses

<table>
<thead>
<tr>
<th>Network</th>
<th>Network purpose</th>
<th>VLAN ID</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.20.3.0/24</td>
<td>IPMI Network</td>
<td></td>
<td>In-band interface</td>
</tr>
<tr>
<td>172.30.4.0/22</td>
<td>Internal Network</td>
<td></td>
<td>bond0</td>
</tr>
<tr>
<td>10.240.37.0/22</td>
<td>External Network</td>
<td>546</td>
<td>eno1.546</td>
</tr>
</tbody>
</table>

Switch Management Addresses
Table 6 lists the management IP addresses for the switches.

Table 6. Switch addresses

<table>
<thead>
<tr>
<th>Network</th>
<th>Network purpose</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lenovo ThinkSystem NE10032 RackSwitch</td>
<td>172.20.4.3</td>
<td>13</td>
</tr>
<tr>
<td>Lenovo ThinkSystem NE10032 RackSwitch</td>
<td>172.20.4.4</td>
<td>14</td>
</tr>
<tr>
<td>Lenovo RackSwitch G7052</td>
<td>172.20.4.1</td>
<td></td>
</tr>
</tbody>
</table>

Server Addresses

Table 7. Server addresses

<table>
<thead>
<tr>
<th>Host name</th>
<th>bond0</th>
<th>eno1.546</th>
<th>In-bound IMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>master1</td>
<td>172.30.4.9</td>
<td></td>
<td>172.20.3.9</td>
</tr>
<tr>
<td>master2</td>
<td>172.30.4.10</td>
<td></td>
<td>172.20.3.10</td>
</tr>
<tr>
<td>master3</td>
<td>172.30.4.11</td>
<td></td>
<td>172.20.3.11</td>
</tr>
<tr>
<td>infra1</td>
<td>172.30.4.7</td>
<td>VIP 10.240.37.132</td>
<td>172.20.3.7</td>
</tr>
<tr>
<td>infra2</td>
<td>172.30.4.8</td>
<td>VIP 10.240.37.132</td>
<td>172.20.3.8</td>
</tr>
<tr>
<td>app1</td>
<td>172.30.4.1</td>
<td></td>
<td>172.20.3.1</td>
</tr>
<tr>
<td>app2</td>
<td>172.30.4.2</td>
<td></td>
<td>172.20.3.2</td>
</tr>
<tr>
<td>app3</td>
<td>172.30.4.3</td>
<td></td>
<td>172.20.3.3</td>
</tr>
<tr>
<td>app4</td>
<td>172.30.4.4</td>
<td></td>
<td>172.20.3.4</td>
</tr>
<tr>
<td>app5</td>
<td>172.30.4.5</td>
<td></td>
<td>172.20.3.5</td>
</tr>
<tr>
<td>app6</td>
<td>172.30.4.6</td>
<td></td>
<td>172.20.3.6</td>
</tr>
<tr>
<td>b01</td>
<td>172.30.4.12</td>
<td>10.240.37.131</td>
<td>172.20.3.12</td>
</tr>
</tbody>
</table>

Table 9. Network connectivity table

<table>
<thead>
<tr>
<th>Device name</th>
<th>Device type</th>
<th>Device role</th>
<th>IMM Switch Port</th>
<th>10G #1 Switch #1 Port</th>
<th>10G #1 Switch #2 Port</th>
<th>IMM</th>
<th>Node Ports</th>
<th>Data Port 1</th>
<th>Data Port 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GbE Switch</td>
<td>Lenovo RackSwitch G7052</td>
<td>Management switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 GbE Switch 2</td>
<td>Lenovo ThinkSystem NE10032 RackSwitch</td>
<td>Data switch 2</td>
<td>14</td>
<td>1/32</td>
<td>1/32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 GbE Switch 1</td>
<td>Lenovo ThinkSystem NE10032 RackSwitch</td>
<td>Data switch 1</td>
<td>13</td>
<td>1/31</td>
<td>1/31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>node12</td>
<td>Lenovo System x3550 M5</td>
<td>Bastion node</td>
<td>12</td>
<td>1/3/4</td>
<td>1/3/4</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node11</td>
<td>Lenovo System x3550 M5</td>
<td>Master node 3</td>
<td>11</td>
<td>1/3/3</td>
<td>1/3/3</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node10</td>
<td>Lenovo System x3550 M5</td>
<td>Master node 2</td>
<td>10</td>
<td>1/3/2</td>
<td>1/3/2</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node09</td>
<td>Lenovo System x3550 M5</td>
<td>Master node 1</td>
<td>9</td>
<td>1/3/1</td>
<td>1/3/1</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node08</td>
<td>Lenovo System x3550 M5</td>
<td>Infra node 2</td>
<td>8</td>
<td>1/2/4</td>
<td>1/2/4</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node07</td>
<td>Lenovo System x3550 M5</td>
<td>Infra node 1</td>
<td>7</td>
<td>1/2/3</td>
<td>1/2/3</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node06</td>
<td>Lenovo System x3550 M5</td>
<td>App node 6</td>
<td>6</td>
<td>1/2/2</td>
<td>1/2/2</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node05</td>
<td>Lenovo System x3550 M5</td>
<td>App node 5</td>
<td>5</td>
<td>1/2/1</td>
<td>1/2/1</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node04</td>
<td>Lenovo System x3550 M5</td>
<td>App node 4</td>
<td>4</td>
<td>1/1/4</td>
<td>1/1/4</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node03</td>
<td>Lenovo System x3550 M5</td>
<td>App node 3</td>
<td>3</td>
<td>1/1/3</td>
<td>1/1/3</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node02</td>
<td>Lenovo System x3550 M5</td>
<td>App node 2</td>
<td>2</td>
<td>1/1/2</td>
<td>1/1/2</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
<tr>
<td>node01</td>
<td>Lenovo System x3550 M5</td>
<td>App node 1</td>
<td>1</td>
<td>1/1/1</td>
<td>1/1/1</td>
<td>Dedicated IMM port</td>
<td>ens1f0</td>
<td>ens1f1</td>
<td></td>
</tr>
</tbody>
</table>
Management Network

For out-of-band management of the servers and initial cluster deployment over the network from the bastion node, use the 1 Gbps management fabric via the Lenovo RackSwitch G7052. The Lenovo System x3550 M5 rack server has a dedicated 1 GbE port for the 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.

In addition to in-band management via IPMI, the Lenovo XClarity Administrator software provides out-of-band management of the servers. Lenovo XClarity Administrator is a centralized systems-management solution. This solution integrates easily with Lenovo System x M5 and X6 rack servers, Lenovo™ Converged HX Series appliances, and Lenovo™ Flex System, providing automated agent-less discovery, monitoring, firmware updates, and configuration management. The Lenovo XClarity software runs as a virtual appliance, meaning it does not require a dedicated physical system to run. The virtual appliance is available for multiple different hypervisors, including Linux KVM, VMware vsphere®, and Microsoft Hyper-V®.

Lenovo™ XClarity® Administrator Software

Lenovo XClarity is a fast, flexible, and scalable hardware systems management application that enables administrators to deploy infrastructure faster and with less effort. The application seamlessly integrates into Lenovo servers, Lenovo Flex System, Lenovo RackSwitch networking, and Lenovo™ S Series storage.

- Reduce the manual effort required to track and comply with user-specified firmware levels throughout the system lifecycle
- Rapidly provision and pre-provision multiple systems using configuration patterns, which contain a single set of defined configuration settings. Predefined UEFI settings can help jumpstart optimal configuration patterns for specific workload environments
- For simpler and faster delivery of systems, the solution uniquely deploys operating systems or hypervisors onto bare metal servers
- Grant users role-based access and authenticate them using the Lenovo XClarity internal LDAP server, an external Microsoft Active Directory® server, or a third-party enterprise single sign-on or multifactor authentication service
- Manage systems from external, higher-level cloud orchestration and IT service-management software tools, making use of available REST APIs
- Forward Simple Network Management Protocol (SNMP) and Syslogs to external event consolidation software tools to aggregate, correlate, and monitor hardware events and runtime issues

![Figure 9. Lenovo XClarity Administrator graphical interface](image)

Lenovo XClarity Administrator features are summarized below.

- Dashboard-driven interface helps administrators find information and complete tasks faster
- Automatically discover Lenovo System x3550 and Lenovo Flex System infrastructure, and gain at-a-glance views of hardware inventory status

![Figure 10. Server overview in Lenovo XClarity Administrator](image)

Figure 11. Policy deployment controls in Lenovo XClarity Administrator
Installing Lenovo XClarity Administrator on the Bastion Node

The steps to install the Lenovo XClarity Administrator software on the bastion node are described below:

1. Download the KVM version of the Lenovo XClarity virtual appliance image from the following site: http://lenovofiles.com/downloads/xclarity-trial. You need to download the Linux qcow2 formatted image for the Lenovo XClarity virtual appliance.

2. Ensure the pre-requisites for installing Lenovo XClarity on Linux KVM hypervisor are configured on the bastion node. Follow the guidance on this page: http://flexsystem.lenovofiles.com/help/topic/com.lenovo.lxca.doc/setup_kvm_installlxca.html?cp=1_6_0_1

3. Create a network configuration file to define the "external" network interface that will be used to access the Lenovo XClarity Administrator. Name this file "eth0_config," and add the following parameters to this file:

   `IPV4_ADDR=<External IP address for xClarity access>`
   `IPV4_NETMASK=<external network subnet mask>`
   `IPV4_GATEWAY=<external network gateway IP>`

4. Following the steps described in the link above, create an ISO image of this file, which will be mounted to the Lenovo XClarity image while being deployed.

5. Create a non-root user account for deploying the Lenovo XClarity image and give it "sudo" access. Then create a bash shell script as shown below:

   ```bash
   #!/bin/bash
   sudo virt-install \
   --name=lxca-1.3.1-74-2 \
   --disk path=/home/user/lnvgy_sw_lxca_74-1.3.1_kvm_x86-6.qcow2,format=qcow2,bus=virtio \
   --graphics vnc,listen=0.0.0.0 \
   --vcpus=2 --ram=8192 \
   --network bridge=virbr0 \
   --network bridge=virbr1 \
   --os-type=generic \
   --arch=x86_64 \
   --cdrom=/home/srihari/boot.iso \
   --noautoconsole
   ```

6. From the non-root account, run the above bash script to deploy the Lenovo XClarity Administrator image. Once the deployment is successful, you should be able to ping the IP address of the Lenovo XClarity Administrator.

7. Log on to the Lenovo XClarity Administrator dashboard. The first time, the Lenovo XClarity setup wizard will run. Follow the steps in the setup wizard to configure the user accounts and other settings.

Once Lenovo XClarity is successfully deployed, you will be able to start managing your hardware. The steps for managing the systems are described in this guide: http://flexsystem.lenovofiles.com/help/topic/com.lenovo.lxca.doc/server_manage.html?cp=1_14

BARE METAL PROVISIONING AND ORCHESTRATION

Extreme Cluster/Cloud Administration Toolkit (xCAT)

xCAT is a tool used to easily manage large numbers of physical servers. Figure 13 shows the architecture and main components of this solution. Lenovo xCAT code can be found on Github at https://github.com/lenovo/lpd-openshift-ra. xCAT key features include:

- Discover hardware and node registration in an internal database
- Operating system deployment
- Switch management
- Remote command execution and power control
- Operating system deployment and configuration
There are various network services necessary to perform operating system deployment over the network. xCAT brings up the network services automatically during installation without any intervention from the system administrator.

**Services used to provide functionality:**

- **DHCP**: DHCP is a protocol that automatically provides IP addresses and simple interface configuration to provide information such as PXE/IPXE server configuration and localization of the bootloader.
- **TFTP**: File transfer protocol that is used to provide the necessary files, download a bootloader, and execute it.
- **HTTP**: This protocol is used to serve unattended files for the operating system installer and post-installation configuration scripts.
- **DNS**: This network service, combined with DHCP, ensures a name configuration for the node servers.
- **Database**: xCAT holds all information about devices as objects. The objects and configuration data are stored in a database. By default, xCAT uses SQLite*, but databases like MySQL, MariaDB, PostgreSQL are also supported here.
- **IPMI**: A module embedded in the hardware server used to perform the out-of-band hardware control (such as IMM, Flexible Service Processor [FSP], and BMC).

**Lenovo Platform Deployer**

Lenovo Platform Deployer (LPD) is an xCAT-based infrastructure configuration and orchestration tool. Some of its key features include:

- **Simplicity**: Automated hardware provisioning
- **Lenovo hardware awareness**: Custom tailored for Lenovo servers based on Intel® processors with IMM2 modules and switches with Lenovo CNOS
- **Portability**: Implemented as a Docker container application
- **All-inclusiveness**: All required dependencies, including xCAT, are embedded in the container
- **Toolset with additional underlying APIs**
- **Extensibility**: Can be used to deploy additional software on top of the host operating system
- **Ansible tie-in**: Generate an Ansible inventory file

**PROVISIONING**

**Network Configuration**

The Lenovo ThinkSystem NE10032 RackSwitch, thanks to CNOS, provides a simple, open, and programmable network infrastructure. This RA makes use of its management capabilities to implement automated network provisioning.

### Automatic Switch Provisioning

This RA contains a submodule provided by Lenovo Modules for Ansible. To obtain the submodule, execute the following commands in the main directory of the RA repository:

```bash
$ git submodule init
$ git submodule update
```

Fill the main inventory file with switch information:

```yaml
[switches]
.x.x.x.x username=SWITCH_ADMIN_USER password=SWITCH_ADMIN_PASSWORD
y.y.y.y username=SWITCH_ADMIN_USER password=SWITCH_ADMIN_PASSWORD
```

Enable VLAG on the switches, if not configured already:

```bash
vlag tier-id 10
vlag isl port-aggregation 1
vlag hlthchk peer-ip neighbor-switch-ip vrf management
vlag enable
```

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

```bash
$ cd src/cnos-configuration/roles/configure-networking
$ env PYTHONPATH=`pwd`/ansible-cnos/library/ ansible-playbook \configure-networking.yaml
```

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

**Manual Switch Configuration**

If you choose to use manual steps, you must perform the following steps on both switches.
Enable VLAG:

- vlag tier-id 10
- vlag isl port-aggregation 1
- vlag hlthchk peer-ip neighbor-switch-ip vrf management
- vlag enable

Configure matching Ethernet interfaces per machine bonding through the aggregation group:

- interface Ethernet1/X/Y
  - bridge-port mode trunk
  - bridge-port trunk native vlan 1001
  - bridge-port trunk allowed vlan 1,1001
  - aggregation-group 32 mode active
  - spanning-tree port type edge

Configure aggregation ports:

- interface port-aggregation32
  - bridge-port mode trunk
  - bridge-port trunk native vlan 1001
  - bridge-port trunk allowed vlan 1,1001

Save the running configuration.

Prepare the Management Switch

Before starting use of the LPD, set up the Lenovo RackSwitch G7052 management switch as the following:

- Cable the server IMM ports to port 1 to 12 of the Lenovo RackSwitch G7052.
- Cable the server’s onboard Ethernet 1 ports to port 23 to 34 of the Lenovo RackSwitch G7052.
- Enable these ports on the Lenovo RackSwitch G7052 and configure them to use VLANs if necessary.
- Cable and configure the uplink ports to the two Lenovo ThinkSystem NE10032 RackSwitch switches.
- Configure the switches with static IP addresses: 172.20.4.1 for the Lenovo RackSwitch G7052, 172.20.4.3 and 172.20.4.4 for the two Lenovo ThinkSystem NE10032 RackSwitch switches.

Downloading Lenovo Platform Deployer code

The source for building the LPD docker container image is available from the Lenovo public github site hosted here: https://github.com/lenovo/lpd-openshift-ra.

You can “git clone” the repository to your management (or deployer) server and follow the instructions on the site above to build your container image.

Install the LPD container on the bastion node by loading the lpd_deployer image. For example:

```
# docker load -i /lpd_deployer-170520
```

After the loading, prepare a configuration file for LPD.

The format is:

```
# first_node last_node os_type (rhela7/rhela7) node_nic node_type (app/infra/master)
1 6 rhela7 1 app
7 8 rhela7 1 infra
9 11 rhela7 1 master
```

Each line specifies a range of server nodes in the test bed in sequential order and indicates the desired operating system type, the intended onboard Ethernet port, and the role of the node, which can be one of the following: application, infrastructure, or master. Save this file to a shared directory that can be mounted with the Docker container. For example, save it to /shared/lpd_config.txt. With the configuration file, run the container by issuing the following command:

```
# docker run -dit -e THINKAGILE_CONFIG_FILE=/shared/lpd_config.txt --stop-signal=RTMIN+3 -v /shared:/shared --net=host --privileged -e "container=docker" --cap-add SYS_ADMIN --security-opt seccomp:unconfined --name=lpd-deployer localhost:5000/lcideployer:lpd_deployer-170520 bash
```

In this example, the LPD is run as a daemon (with –d) with a predefined configuration file, /shared/lpd_config.txt; whereas the directory, /shared, is used as a shared directory between the host and the container.

Check the installation is successful with the following commands:

```
# docker exec -it lpd-deployer lpdeploy -h
```

The Lenovo Platform Deployer is a management tool which provides a simple interface to configure and deploy OS/Apps to a Lenovo ThinkAgile™ platform.

Usage: lpdeploy [-c <file>] Configure and deploy the target using <file> as input if specified. Otherwise, use the file in the THINKAGILE_CONFIG environment variable.  
-i <interface> Use the network interface on the Deployer host to configure  
-h/-? Print this help  
-v Display the current Deployer version  
-l List/generate an ansible inventory file to /shared/lpd_inventory.txt  
-x <cmd> Execute a (xcat) command  
-s Show the setup info from the previous run  
-p <passwd> Set a default password

Examples:

```
>lpdeploy
>lpdeploy -c /shared/lpd_config.txt
>lpdeploy -i eno2
>lpdeploy -x "lsdef hosts"

# docker exec -it lpd-deployer lpdeploy -v
Lenovo Platform Deployer version 0.51.osp
```
Before provisioning the hardware and installing the host operating system, make sure the following checklist is ready:

- Check that the servers are reset to the factory defaults.
- Check if the desired virtual drive is created on each server (for example, a virtual drive consisting of two 480 GB drives should be created on the infrastructure nodes as RAID1).
- Make sure the bastion node can connect to the switches, and gather the server IMM link local addresses (e.g., fe80::0a94:efff:fe25:b198). These addresses can be found on either the physical server tags or from the switches by looking into the MAC address table.
- Test the server IMM port connections by issuing the following (assuming the server is reachable via eno1):

  # ping6 fe80::0a94:efff:fe25:b198%eno1

When the servers and switches are ready, start the deployment by issuing:

  # docker exec –it lpd-deployer lpdeploy

Figure 14 shows the LPD workflow.

The entire provisioning and orchestration process takes about 30 minutes for 11 nodes. The LPD logs its orchestration process in a tmux window, which can be reviewed using the following command:

  # docker exec -it lpd_deployer lpdeploy "tmux a"

This displays a tmux screen similar to the following:

```
# docker exec -it lpd_deployer lpdeploy --s
Appliance MAC Addresses:
[50/1993]
infral-mgmt: MAC Address 1: 08:94:ef:25:b1:93
infral-mgmt: MAC Address 3: 08:94:ef:25:b1:95
app6-mgmt: MAC Address 1: 08:94:ef:25:b2:bb
app6-mgmt: MAC Address 2: 08:94:ef:25:b2:bc
app6-mgmt: MAC Address 3: 08:94:ef:25:b2:bd
app6-mgmt: MAC Address 4: 08:94:ef:25:b2:be
app2-mgmt: MAC Address 1: 08:94:ef:25:aa:db
app2-mgmt: MAC Address 2: 08:94:ef:25:aa:dc
app2-mgmt: MAC Address 3: 08:94:ef:25:aa:dd
app2-mgmt: MAC Address 4: 08:94:ef:25:aa:de
app5-mgmt: MAC Address 1: 08:94:ef:25:b8:2b
app5-mgmt: MAC Address 2: 08:94:ef:25:b8:2c
app5-mgmt: MAC Address 3: 08:94:ef:25:b8:2d
app5-mgmt: MAC Address 4: 08:94:ef:25:b8:2e
app1-mgmt: MAC Address 4: 08:94:ef:25:a9:16
master3-mgmt: MAC Address 4: 08:94:ef:25:a9:16
master1-mgmt: MAC Address 1: 08:94:ef:25:a3:5b
master1-mgmt: MAC Address 2: 08:94:ef:25:a3:5c
master1-mgmt: MAC Address 3: 08:94:ef:25:a3:5d
master1-mgmt: MAC Address 4: 08:94:ef:25:a3:5e
master2-mgmt: MAC Address 1: 08:94:ef:25:a2:8b
```

Select a network interface connecting to the test bed and start the installation. After the process completes, retrieve the successful result, which includes a list of server MAC and host addresses, using the following command:

  # docker exec –it lpd_deployer lpdeploy –s

Appliance MAC Addresses:
[50/1993]
infral-mgmt: MAC Address 1: 08:94:ef:25:b1:93
infral-mgmt: MAC Address 3: 08:94:ef:25:b1:95
app6-mgmt: MAC Address 1: 08:94:ef:25:b2:bb
app6-mgmt: MAC Address 2: 08:94:ef:25:b2:bc
app6-mgmt: MAC Address 3: 08:94:ef:25:b2:bd
app6-mgmt: MAC Address 4: 08:94:ef:25:b2:be
app2-mgmt: MAC Address 1: 08:94:ef:25:aa:db
app2-mgmt: MAC Address 2: 08:94:ef:25:aa:dc
app2-mgmt: MAC Address 3: 08:94:ef:25:aa:dd
app2-mgmt: MAC Address 4: 08:94:ef:25:aa:de
app5-mgmt: MAC Address 1: 08:94:ef:25:b8:2b
app5-mgmt: MAC Address 2: 08:94:ef:25:b8:2c
app5-mgmt: MAC Address 3: 08:94:ef:25:b8:2d
app5-mgmt: MAC Address 4: 08:94:ef:25:b8:2e
app1-mgmt: MAC Address 4: 08:94:ef:25:a9:16
master3-mgmt: MAC Address 4: 08:94:ef:25:a9:16
master1-mgmt: MAC Address 1: 08:94:ef:25:a3:5b
master1-mgmt: MAC Address 2: 08:94:ef:25:a3:5c
master1-mgmt: MAC Address 3: 08:94:ef:25:a3:5d
master1-mgmt: MAC Address 4: 08:94:ef:25:a3:5e
master2-mgmt: MAC Address 1: 08:94:ef:25:a2:8b

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

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 `nfs` (for Docker Registry persistent storage), `lb` (for load balancing in multi-master clusters), and `local` (which specifies the bastion node).

```bash
[OSEv3:children]
nodes
masters
nfs
etc
lb
local
```

```bash
[OSEv3:vars]
anible_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': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/users.htpasswd'}
]
os_sdn_network_plugin_name=redhat/openshift-ovs-multitenant
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_volume_size=300G
```
This section describes global cluster parameters. Parameter `openshift_master_cluster_method` specifies the load balancing method in a multi-master environment. With the native value, there will be 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 way for authentication of OpenShift users. In this example, this parameter is based on htpasswd files stored in the OpenShift configuration directory. However, you can use many other authentication methods like LDAP, Keystone*, or GitHub* accounts. `os_sdn_network_plugin_name` specifies the SDN Open vSwitch plugin used in environment. In this RA, `redhat/openshift-ovs-multitenant` provides isolation between OpenShift projects on the network level. The last two sections specify the storage backend type and its size for Docker Registry. In this solution, Docker Registry uses NFS server for Docker image storage.

This section describes which servers act as OpenShift nodes. In this RA, 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 servers. However, no user application should be deployed on these servers because of the `openshift_schedulable=False` parameter. All other nodes have scheduling enabled. In this RA, all node components are implemented as containers, which is determined by the `containerized=True` parameter.

(Continued)
In this section, the hosts that will run NFS server are specified. NFS server is used as a storage backend for Docker Registry. In OpenShift, NFS server has to be installed on a host with the RHEL operating system (that is, an infrastructure node). In this RA, the first NFS server uses a special partition with the /export mount point. This partition is replicated and backed up by DRBD to a secondary NFS server node. In case of a failure of the first node, the DRBD partition can easily switch into the secondary node with no data loss.

```
[etcd]
etcd1.ocp.example.local containerized=True openshift _
ip=172.30.4.9 openshift _ hostname=etcd1.ocp.example.local
etcd2.ocp.example.local containerized=True openshift _
ip=172.30.4.10 openshift _ hostname=etcd2.ocp.example.local
etcd3.ocp.example.local containerized=True openshift _
ip=172.30.4.11 openshift _ hostname=etcd3.ocp.example.local
```

This section describes hosts that will run etcd instances. In this RA, 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 also recommended to have an odd number of etcd instances in a cluster.

```
[lb]
lb1.ocp.example.local openshift _ hostname=lb1.ocp.example.local openshift _
ip=172.30.4.7
lb2.ocp.example.local openshift _ hostname=lb2.ocp.example.local openshift _
ip=172.30.4.8
```

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

**Node Preparation**

When operating system deployment is finished, the nodes must be prepared for OpenShift installation. Perform the following preliminary steps: prepare an openshift account and exchange SSH keys across all nodes, attach software licenses, install and configure the DNS service, install additional packages, and configure Docker Engine. All tasks can be executed either automatically, using Ansible playbooks available at https://github.com/intel/openshift-container-architecture, or manually.

**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.

Based on information from the operating system deployment, prepare a hosts file. Refer to the example in the Appendix A, and place it in the location /etc/ansible/hosts. After that, clone the Git repository:

```
$ git clone https://github.com/intel/openshift-container-architecture/
$ cd ra-redhat-openshift/src/prerequisitesa
```

In the 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>

Start the playbook by entering the following command:

```
$ ansible-playbook nodes_setup.yaml -k
```

**Manual Steps**

If desired, you can omit automatic prerequisites installation and perform manual steps instead. In order to do so on each node, the openshift user must be created with proper SSH keys.

```
$ sudo useradd -m -G root openshift
```

OpenShift Container Platform Ansible installation requires additional privileges provided through sudo:

```
$ echo openshift ALL=(root) NOPASSWD: ALL | sudo tee /etc/sudoers.d/openshift
```
SSH keys have to be generated on the bastion node and distributed across the cluster:

```bash
$ ssh-keygen
$ ssh-copy-id -i ~/.ssh/id_rsa.pub $host
```

All nodes must be registered with Red Hat Subscription Manager and have an active OpenShift Container Platform subscription attached.

```bash
$ sudo subscription-manager register --username=<username> --password=<password>
```

Available subscriptions can be listed with the following command:

```bash
$ sudo subscription-manager list --available --matches 'OpenShift'
```

Next, attach an OpenShift Container Platform subscription from the available license pool:

```bash
$ sudo subscription-manager attach --pool=<pool_id>
```

To ensure proper software versions, only specific repositories should be enabled:

```bash
$ sudo subscription-manager repos --disable="*"
$ sudo subscription-manager repos \ 
   --enable="rhel-7-server-rpms" \ 
   --enable="rhel-7-server-extras-rpms" \ 
   --enable="rhel-7-server-ose-3.5-rpms"
```

By default, RHEL uses a DNS server. It must be configured to reflect network IP space and the chosen domain. The configuration file `/etc/dnsmasq.conf` must contain the following lines:

```conf
no-hosts
conf-dir=/etc/dnsmasq.d
```

The next step is to create a DNS configuration with domain-to-IP mappings. Below is the DNS file used in this RA. It should be placed in `/etc/dnsmasq.d/hosts`.

```conf
# [master/worker private ip]
address=/master1.ocp.example.local/master1.ocp.example.local/172.30.4.9
address=/master2.ocp.example.local/master2.ocp.example.local/172.30.4.10
address=/master3.ocp.example.local/master3.ocp.example.local/172.30.4.11
address=/infra1.ocp.example.local/infra1.ocp.example.local/172.30.4.7
address=/infra2.ocp.example.local/infra2.ocp.example.local/172.30.4.8
address=/app1.ocp.example.local/app1.ocp.example.local/172.30.4.1
address=/app2.ocp.example.local/app2.ocp.example.local/172.30.4.2
address=/app3.ocp.example.local/app3.ocp.example.local/172.30.4.3
address=/app4.ocp.example.local/app4.ocp.example.local/172.30.4.4
address=/app5.ocp.example.local/app5.ocp.example.local/172.30.4.5
address=/app6.ocp.example.local/app6.ocp.example.local/172.30.4.6
address=/etcd1.ocp.example.local/etcd1.ocp.example.local/172.30.4.9
address=/etcd2.ocp.example.local/etcd2.ocp.example.local/172.30.4.10
address=/etcd3.ocp.example.local/etcd3.ocp.example.local/172.30.4.11
address=/lb1.ocp.example.local/lb1.ocp.example.local/172.30.4.7
address=/lb2.ocp.example.local/lb2.ocp.example.local/172.30.4.8
address=/nfs1.ocp.example.local/nfs1.ocp.example.local/172.30.4.7
address=/nfs2.ocp.example.local/nfs2.ocp.example.local/172.30.4.8
# [lb private ip]
host-record=ocp.example.local,172.30.4.132
# [lb public ip]
address=/ocp.example.com/10.240.37.132
```

All internal interfaces should be configured in bonds. This step should be done during operating system, provisioning. However, manual configuration might be necessary. Below is an example of a configuration file for the appropriate interface on one of the application nodes:

```conf
DEVICE=bond0
NAME=bond0
TYPE=Bond
BOOTPROTO=None
BONDING_MASTER=yes
BONDING_OPTS="miimon=1000 mode=4"
ONBOOT=yes
IPADDR=172.30.4.1
PREFIX=16
DEFROUTE=yes
GATEWAY=172.30.4.12
DNS1=172.30.4.12
```

Reference Architecture | Deploying Red Hat® OpenShift® Container Platform 3.5 on Lenovo™ System x3550 M5 Rack Servers
Key options:
GATEWAY=172.30.4.12—set the gateway address for the host (bastion node)
DNS1=172.30.4.12—set the DNS server address for the host (bastion node)

Those options must be configured only on one of the interfaces.
Nodes with RHEL should have proper packages installed. Run those commands on the bastion and infrastructure nodes.

```bash
$ sudo yum -y install wget git net-tools bind-utils
$ sudo yum -y install atomic-openshift-excluder atomic-openshift-docker-excluder
$ sudo yum -y install docker
$ sudo yum update
```

On the bastion node, install the packages required by OpenShift installer:

```bash
$ sudo yum -y install atomic-openshift-utils
```

On RHEL Atomic Host systems, ensure that they are up to date by upgrading to the latest RHEL Atomic Host tree if one is available:

```bash
$ sudo atomic host upgrade
```

After the package installation is complete, verify that that version matches 1.12:

```bash
$ docker version
```

Edit the `/etc/sysconfig/docker` file and add `--insecure-registry 172.30.0.0/16` to the OPTIONS parameter. This causes Docker to accept insecure registries from the internal OpenShift Container Platform network.

```
OPTIONS='--selinux-enabled --insecure-registry 172.30.0.0/16'
```

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

This RA uses HAProxy instances (in both flavors), which are installed on both infra nodes. In conjunction with floating IP addressed provided by Keepalived, a single point of failure is eliminated. Installation and configuration can be performed manually or through a single command (using an Ansible playbook).

**Automatic Keepalived Deployment**
The following variables must be defined in the Ansible inventory:

```bash
external_interface=eno1
external_vlan=546
internal_interface=bond0
openshift_master_cluster_ip=172.30.4.132
openshift_master_cluster_public_ip=10.240.37.132
```

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

```bash
$ su openshift
$ ansible-playbook
ra-redhat-openshift/src/keepalived-multimaster/keepalived.yaml
```

**Manual Keepalived Deployment**
In order to omit automatic deployment and install Keepalived manually, enter the following command on both infra nodes:

```bash
$ sudo yum -y install keepalived
```

On the first infra node, paste the following configuration in `/etc/keepalived/keepalived.conf`, changing the auth_pass values to random passwords:
On the second `infra` node, paste the following configuration in `/etc/keepalived/keepalived.conf`, changing `auth_pass` to match the previously generated passwords:

```plaintext
 ALLOW Virtual Router Redundancy Protocol (VRRP) traffic on all interfaces:

$ sudo iptables -I INPUT -p vrrp -j ACCEPT

Enable and start Keepalived on both `infra` nodes:

$ sudo systemctl enable keepalived
$ sudo systemctl start keepalived

**NFS + DRBD**

In this RA, DRBD provides backup for the NFS storage backend, where private Docker images are stored. DRBD is not a part of RHEL repositories, but it is supported and certified on RHEL 5/6/7 systems by LINBIT®. Also, DRBD is not a part of the core OpenShift Container Platform architecture, and its installation is optional. However, DRBD is an important component of this solution—it removes a single point of failure for NFS-based Docker Registry storage.

**Automatic DRBD Deployment**

DRBD can be deployed automatically with use of the Ansible playbooks. It will be installed on nodes that are listed in `[nfs]` section in the inventory file (refer to Appendix A). In order to install DRBD automatically, perform the following steps.

(Continued)
Manual DRBD Deployment

If you prefer, you can install DRBD and configure it manually. DRBD is installed on two OpenShift infra nodes. The first acts as an active DRBD instance, and the second acts as a backup DRBD instance. In order to install and configure DRBD on infra nodes, firstly, you must create an LVM partition on both of them. You can do this during operating system provisioning or later with the following command:

```bash
sudo lvcreate --name drbd_registry --size 300G vg00
```

Next, DRBD needs to be installed on both nodes.

```bash
sudo rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-elrepo.org
sudo yum -y install drbd84-utils kmod-drbd84

Create a DRBD config file on both nodes in `/etc/drbd.d/nfs.res`.

```
resource nfs {
  protocol C;
  on infra1 {
    device /dev/drbd0;
    disk /dev/mapper/rhel_infrac-drbd_registry;
    address 172.30.4.7:7788;
    meta-disk internal;
  }
  on infra2 {
    device /dev/drbd0;
    disk /dev/mapper/rhel_infra2-drbd_registry;
    address 172.30.4.8:7788;
    meta-disk internal;
  }
}
```

Run the following commands on both nodes:

```bash
sudo drbdadm create-md nfs
sudo drbdadm up nfs
sudo drbdadm primary nfs --force (only on infra1 node)
sudo drbdadm secondary nfs (only on infra2 node)
sudo iptables -I INPUT -p tcp --dport 7788 -j ACCEPT
sudo systemctl start drbd
sudo systemctl enable drbd

Create a file system and mount a DRBD partition on the primary node (infra1).

```bash
sudo mkfs.xfs /dev/drbd0
mkdir /exports
sudo mount /dev/drbd0 /exports
```

All Docker images will now be stored on the DRBD partition mounted to the `/exports` mount point on the first OpenShift infrastructure node. All data from this partition will be backed up on a second infrastructure node, and, in case of a failure on the primary node, you can switch to the backup DRBD node (infra2).

**Disaster Recovery**

In case of primary DRBD node failure, you can switch Docker Registry and NFS services to make use of a secondary DRBD node. If the primary infrastructure (DRBD) node fails, perform the following steps to switch into the secondary NFS server for Docker Registry.

Log on to the infra2 node from the bastion node, enable DRBD in primary mode, and then mount the partition.

```bash
$ ssh infra2.ocp.example.local
drbdadm primary nfs
$ sudo mount /dev/drbd0 /exports
```

```bash
$ ssh master1.ocp.example.local
$ oc edit pv registry-volume

Change the following section:
nfs:
  path: /exports/registry
  server: nfs1.ocp.example.local

Into:
  nfs:
    path: /exports/registry
    server: nfs2.ocp.example.local
```
The last step is to delete Docker Registry pods so that the deployment config will run them again with new configs and binded volumes.

$ oc get pods | grep registry
  docker-registry-1-7h4k4    1/1     Running   0
  37m
  docker-registry-1-7rzjg    1/1     Running   0
  37m

$ oc delete pod docker-registry-1-7h4k4
$ oc delete pod docker-registry-1-7rzjg

At this time, Docker Registry deployment config should start new pods with new volumes binded from secondary NFS (DRBD) server. All Docker images that were available on the primary node should also be available on the secondary node now. When the primary infrastructure node is back online, you can perform reverse steps in order to switch to this node for a Docker Registry NFS storage backend.

**OPENSHIFT CONTAINER PLATFORM DEPLOYMENT**

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, so the OpenShift Container Platform environment will be set up. If needed, you can easily uninstall the environment with the following command:

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

When installation completes, you must create user credentials. In order to do this, run the following commands:

```
$ sudo yum install httpd-tools
$ touch users.htpasswd
$ htpasswd -n <user_name> >> users.htpasswd
```

You should execute the `htpasswd` command multiple times to create multiple user accounts. Propagate this file to every OpenShift master node, into the `/etc/origin/master/` directory, and then restart the API services on each of them.

```
$ sudo systemctl restart atomic-openshift-master-api
```

Users can now log on to the OpenShift Container Platform web panel available at https://ocp.example.com:8443.

**Deployment Validation**

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

```
$ ssh master1.ocp.example.local
$ oc get nodes
```

```bash
NAME                                      STATUS                        AGE
master1.ocp.example.local                 Ready,SchedulingDisabled      5m
master2.ocp.example.local                 Ready,SchedulingDisabled      5m
master3.ocp.example.local                 Ready,SchedulingDisabled      5m
infra1.ocp.example.local                  Ready                         5m
infra2.ocp.example.local                  Ready                         5m
app1.ocp.example.local                    Ready                         5m
app2.ocp.example.local                    Ready                         5m
app3.ocp.example.local                    Ready                         5m
app4.ocp.example.local                    Ready                         5m
app5.ocp.example.local                    Ready                         5m
app6.ocp.example.local                    Ready                         5m
```

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.

```
$ 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 c30519ff3c57613c 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
```

You can use the above 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.
Type the preceding command to verify the infrastructure node components of the OpenShift Container Platform cluster. It should result with a list of pods that run Docker Registry and router services, and they all should have a Running status.

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

**OPENSHIFT CONTAINER PLATFORM SCALING**

OpenShift Container Platform is a highly scalable and elastic platform that helps users handle a growing need for compute resources in their environments. It offers functions for easy scale up of OpenShift master and node components.

**OpenShift Node Scale Up**

OpenShift nodes can be scaled up when a cluster is deployed with the quick installation method or the advanced installation method. In order to add new OpenShift node hosts to an existing cluster, there should be an additional group, new_nodes, added to its inventory.

```yaml
[OSEv3:children]
nodes
masters
nfs
cored
lb
new_nodes
```

This group should also provide a definition of all new hosts that will be assigned to an existing cluster. The following example shows the inventory for two additional OpenShift nodes.

```yaml
[new_nodes]
app7.ocp.example.local containerized=True openshift_scheduled=True
app8.ocp.example.local containerized=True openshift_scheduled=True
```

You must configure the DNS server with new node hostnames so they will be available from the bastion node. Also, their keys should already be added to the known_hosts file on the bastion node. The last step is to run the OpenShift Ansible scale up playbook. After it’s done, new OpenShift nodes should be added and configured in the cluster environment.

```bash
$ su openshift
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/byo/openShift-node/scaleup.yml
```

When the scaling process is finished without failures, remove the new_nodes group from the inventory file and add the newly created OpenShift node’s definition to the nodes group.

```yaml
[nodes]
master1.ocp.example.local containerized=True openshift_scheduled=False
master2.ocp.example.local containerized=True openshift_scheduled=False
master3.ocp.example.local containerized=True openshift_scheduled=False
infra1.ocp.example.local containerized=True openshift_node_labels="{'region': 'infra'}"
infra2.ocp.example.local containerized=True openshift_node_labels="{'region': 'infra'}"
app1.ocp.example.local containerized=True openshift_scheduled=True
app2.ocp.example.local containerized=True openshift_scheduled=True
app3.ocp.example.local containerized=True openshift_scheduled=True
app4.ocp.example.local containerized=True openshift_scheduled=True
app5.ocp.example.local containerized=True openshift_scheduled=True
app6.ocp.example.local containerized=True openshift_scheduled=True
app7.ocp.example.local containerized=True openshift_scheduled=True
app8.ocp.example.local containerized=True openshift_scheduled=True
```

**OpenShift Master Scale Up**

When the OpenShift Container Platform cluster was deployed with an advanced installation method, its control plane can be easily scaled up. In order to add new OpenShift master hosts to an existing cluster, there should be an additional new_masters group added to its inventory. Every OpenShift master also has OpenShift node components installed. This means that the new_nodes group also needs to be added to an inventory file.
Also, those groups should provide a definition of all new hosts that will be assigned to an existing cluster. The following example shows the inventory for two additional OpenShift masters.

```
[masters]
master4.ocp.example.local containerized=True
master5.ocp.example.local containerized=True

[new_masters]
master4.ocp.example.local containerized=True
master5.ocp.example.local containerized=True

[nodes]
master4.ocp.example.local containerized=True openshift_schedulable=False
master5.ocp.example.local containerized=True openshift_schedulable=False

infra1.ocp.example.local containerized=True openshift_node_labels="{'region':'infra'}"
infra2.ocp.example.local containerized=True openshift_node_labels="{'region':'infra'}"
app1.ocp.example.local containerized=True openshift_schedulable=True
app2.ocp.example.local containerized=True openshift_schedulable=True
app3.ocp.example.local containerized=True openshift_schedulable=True
app4.ocp.example.local containerized=True openshift_schedulable=True
app5.ocp.example.local containerized=True openshift_schedulable=True
app6.ocp.example.local containerized=True openshift_schedulable=True
```

You must configure the DNS server with new master hostnames so they will be available from the bastion node. Their keys should also already be added to the known_hosts file on the bastion node. The last step is to run OpenShift Ansible scale up playbook. After it’s done, new OpenShift masters should be added and configured in the cluster environment.

```
$ su openshift
$ ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/byo/openshift-master/scaleup.yml
```

When the scaling process is finished without failures, remove the new_nodes and new_masters groups from the inventory file and add the newly created OpenShift node and OpenShift master definitions to their proper groups so they will map to a new cluster topology.

**Etcd Cluster Scale Up**

Although it is easy and fast to scale OpenShift master and OpenShift node components, it can be problematic to scale up an etcd cluster with OpenShift Ansible playbooks. However, when needed, an etcd cluster can be scaled up with the Ansible playbook available at https://github.com/intel/openshift-container-architecture. In order to add new etcd members to an existing etcd cluster, add a new_etcd group to the OpenShift inventory file.
This group should provide a definition of all new hosts that will be assigned to an existing cluster. The following example shows the inventory for two additional etcd members:

```
[new_etcd]
etcd4.ocp.example.local containerized=True
netcd5.ocp.example.local containerized=True
```

You must configure the DNS server with new etcd hostnames so they will be available from bastion node. Also, their keys should already be added to `known_hosts` file on the bastion node. The last step is to run the Ansible scale up playbook available at https://github.com/intel/openshift-container-architecture. After it’s done, new etcd members should be added and configured in the cluster environment.

```
$ su openshift
$ ansible-playbook src/etcd-scaling/scale.yml
```

When the scaling process is finished without failures, remove the `new_etcd` group from the inventory file and add the newly created etcd member definitions to their proper groups so they will map to a new cluster topology.

```
[etcd]
etcd1.ocp.example.local containerized=True
netcd2.ocp.example.local containerized=True
netcd3.ocp.example.local containerized=True
netcd4.ocp.example.local containerized=True
netcd5.ocp.example.local containerized=True
```

**SUMMARY AND CONCLUSIONS**

Red Hat solutions involving the Red Hat OpenShift Container Platform 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 RA covered the process of provisioning and deploying a highly available OpenShift Container Platform cluster on a private cloud environment with use of Lenovo System x3550 M5 rack servers powered by Intel Xeon processor E5-2600 v4 family.

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

---


Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit intel.com/benchmarks.

Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Red Hat, Ansible, OpenShift, the OpenShift logo, and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

Lenovo, the Lenovo system, Lenovo RackSwitch, ThinkSystem, and XClarity are trademarks of Lenovo in the United States, other countries, or both.

Intel, the Intel logo, Intel Atom, and Xeon are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

© 2017 Intel Corporation.
APPENDIX A: OCP DEPLOYMENT INVENTORY FILE

```
[OSEv3:children]
masters
nodes
nfs
etcd
lb
local

[OSEv3:vars]
anible_ssh_user=openshift
ansible_become=true
openshift_use_dnsmasq=false
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
deployment_type=openshift-enterprise
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_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_volume_size=300Gi
local_dns=172.30.4.12
external_interface=enol
external_vlan=546
internal_interface=bond0
openshift_master_cluster_ip=172.30.4.132
openshift_master_cluster_public_ip=10.240.37.132
openshift_master_portal_net=10.0.0.0/16
openshift_release=v3.5
rhom_subscription_user=RED_HAT_PORTAL_LOGIN
rhom_subscription_pass=RED_HAT_PORTAL_PASSWORD

[local]
127.0.0.1

[masters]
master1.ocp.example.local containerized=True openshift_scheduled=False openshift_ip=172.30.4.9 openshift_hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift_scheduled=False openshift_ip=172.30.4.10 openshift_hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True openshift_scheduled=False openshift_ip=172.30.4.11 openshift_hostname=master3.ocp.example.local

[nfs]
fns1.ocp.example.local openshift_hostname=nfs1.ocp.example.local openshift_ip=172.30.4.7
fns2.ocp.example.local openshift_hostname=nfs2.ocp.example.local openshift_ip=172.30.4.8

[etcd]
etcd1.ocp.example.local containerized=True openshift_scheduled=True openshift_ip=172.30.4.9 openshift_hostname=etcd1.ocp.example.local
etcd2.ocp.example.local containerized=True openshift_scheduled=True openshift_ip=172.30.4.10 openshift_hostname=etcd2.ocp.example.local
etcd3.ocp.example.local containerized=True openshift_scheduled=True openshift_ip=172.30.4.11 openshift_hostname=etcd3.ocp.example.local

[lb]
lb1.ocp.example.local openshift_hostname=lb1.ocp.example.local openshift_ip=172.30.4.7
lb2.ocp.example.local openshift_hostname=lb2.ocp.example.local openshift_ip=172.30.4.8
```

(Continued)