REFERENCE ARCHITECTURE

Intel® Builders
Intel® Xeon® Processors

Deploying Red Hat® OpenShift® Container Platform 3.5 on Lenovo™ System x3550 M5 Rack Servers

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. 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.

INTEL, LENOVO, RED HAT, AND OPENSHIFT

OpenShift Container Platform 3.5 by Red Hat is built around a core of application containers powered by Docker, with orchestration and management provided by Kubernetes, on a foundation of Red Hat® Enterprise Linux* Atomic Host. It provides many enterprise-ready features, like enhanced security features, multitenancy, simplified application deployment, and continuous integration/continuous deployment tools. With Lenovo™ servers and technologies, provisioning and managing the OpenShift Container Platform 3.5 infrastructure becomes practically effortless and produces a resilient solution.

This document describes the system architecture for the OpenShift platform based on Lenovo™ System x3550 M5 servers and network switches. These servers are powered by the Intel® Xeon® processor E5-2600 v4 product family, which provides more than 20 percent more cores than the Intel Xeon processor E5-2600 v3 product family, supports faster memory, and includes technologies for accelerating specific workloads. This document provides detail of the hardware requirements to support various OpenShift node roles and the corresponding configuration of the systems. It also describes the network architecture and details for the switch configurations. The hardware bill of materials for all required components to assemble the OpenShift cluster is provided, in addition to the rack-level design and power configuration. The automation logic for deploying the hardware infrastructure in preparation for the OpenShift implementation is also described.

Authors
Srihari Angaluri
Joe Carvalho
Ta Ming Chen
Dariusz Komła
Łukasz Łuczaj
Jose Palafox
Markesha Parker
Łukasz Sztachański

Contents

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 1. Software versions

Component	Versions
Red Hat Enterprise Linux	7.3
Red Hat Enterprise Linux Atomic Host	7.3.3
OpenShift Container Platform	3.5
Docker	1.12.6
Ansible	2.2.1.0
Etcd	3.1.0
Open vSwitch	2.6.1
HAProxy	1.5.18
Keepalived	1.2.13
DRBD	8.9.8

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.

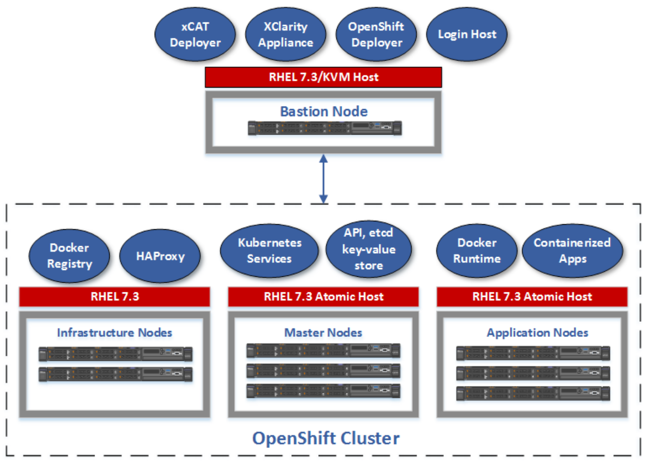


Figure 1. OpenShift Node Roles

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.

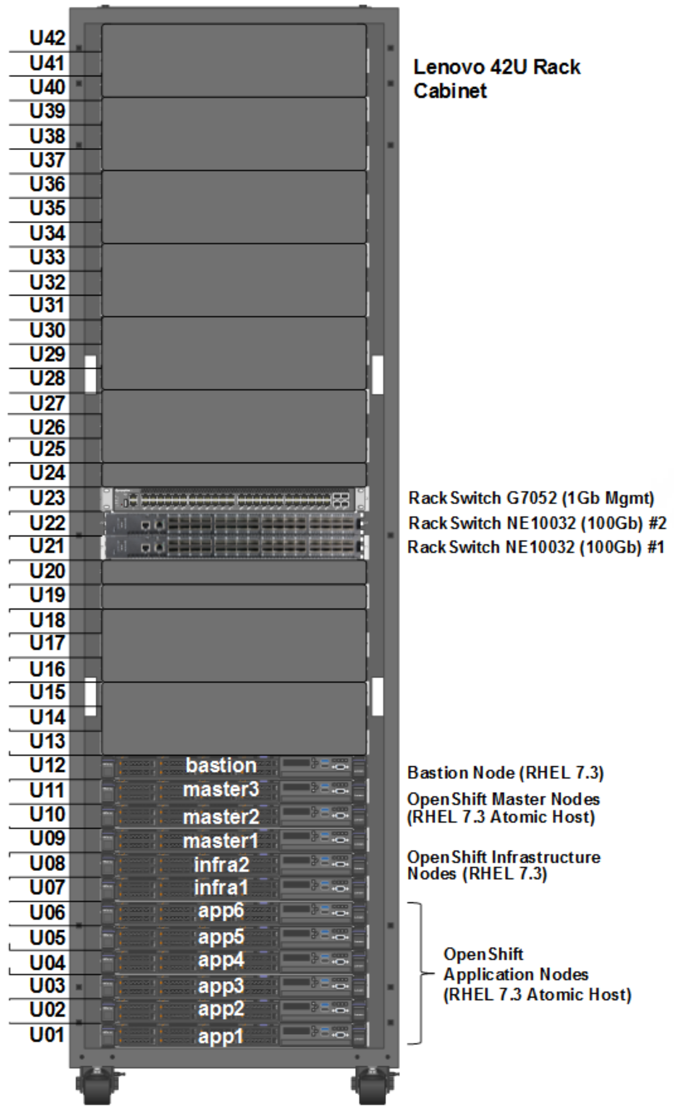


Figure 2. Rack diagram for the Lenovo and OpenShift RA

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.

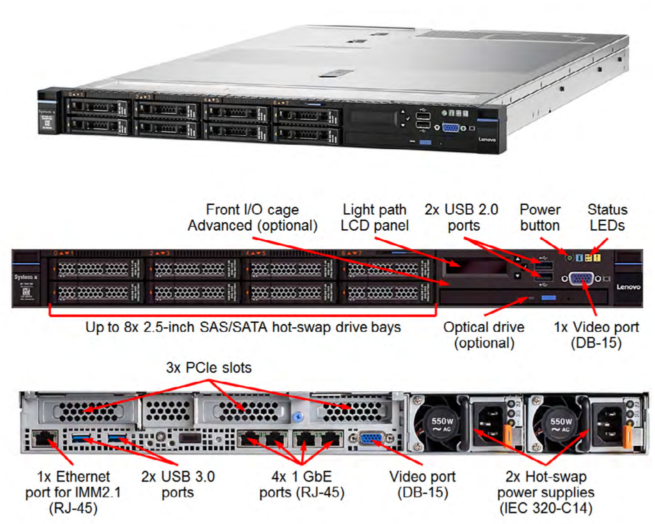


Figure 3. The Lenovo System x3550 M5 rack server

More information on the Lenovo System x3550 M5 server is available here: https://lenovopress.com/lp0067-lenovo-system-x3550-m5-machine-type-8869

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[1]
- **Intel® Transactional Synchronization Extensions (Intel® TSX)** for high performance of multi-threaded workloads

- **Intel® Advanced Vector Extensions 2.0 (Intel® AVX2)** for high performance of mixed workloads
- **Enhanced Intel® Data Direct I/O (Intel® DDIO)** for fast access to critical data
- **Intel® Virtual Machine Control Structure (Intel VMCS)** for enhanced virtualization
- **Intel® Resource Director Technology (Intel RDT)** for smarter resource orchestration

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.

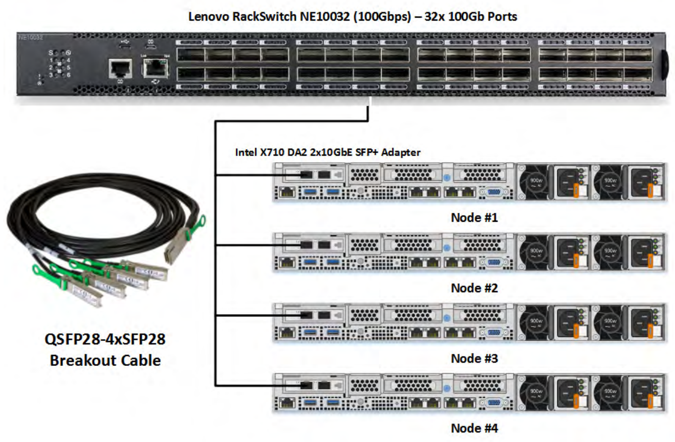

Lenovo RackSwitch NE10032 (100Gbps) – 32x 100Gb Ports
Intel X710 DA2 2x10GbE SFP+ Adapter
Node #1
Node #2
Node #3
Node #4
QSFP28-4xSFP28 Breakout Cable

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

OpenShift Role	Qty.	Platform	Configuration
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

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

Quantity	Description	MTM	FC	Notes
1x Rack Enclosure	42U 1,200 mm Deep Dynamic Rack (six sidewall compartments)	9363-4PX	7649	https://lenovopress.com/lp0658-lenovo-rack-cabinet-reference
1x Management Switch	Lenovo RackSwitch G7052 (1Gbps)	7159-HCT	AT0A	https://lenovopress.com/tips1269-lenovo-rackswitch-g7052
2x Cluster Traffic Switches	Lenovo ThinkSystem NE10032 RackSwitch (100 Gbps)			https://lenovopress.com/lp0609.pdf
• 1x Bastion • 2x Infrastructure • 3x Management • 6x Application	Lenovo System x3550 M5 Server	8869		https://lenovopress.com/lp0067-lenovo-system-x3550-m5-machine-type-8869
2-4	Rack power distribution units (PDUs)			Refer to Table 4.

Table 4. Lenovo power distribution units (PDUs)

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

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/lnvo-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.

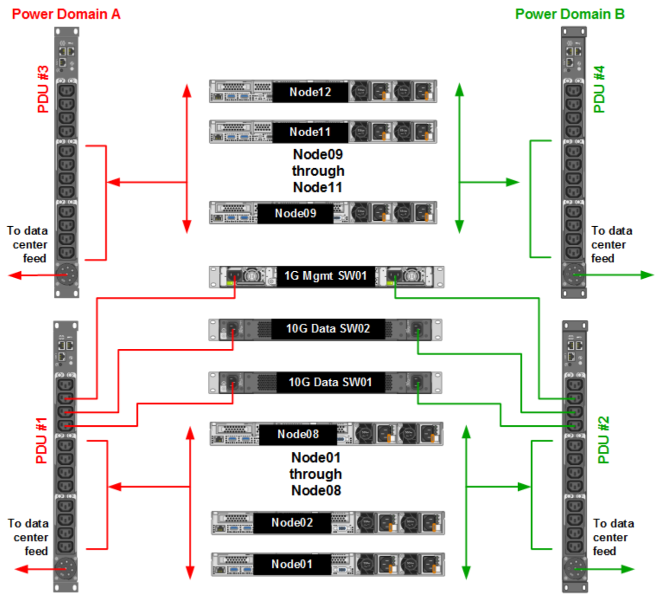


Figure 5. Rack power diagram

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.

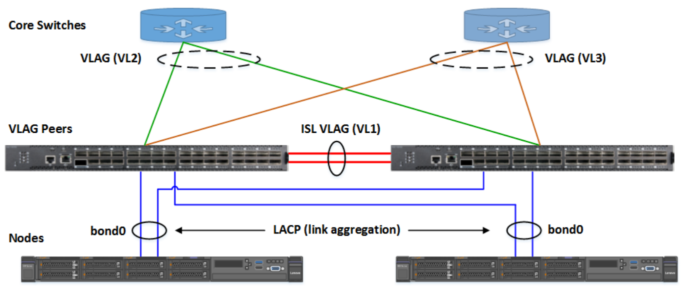


Figure 6. Virtual Link Aggregation (VLAG)

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).

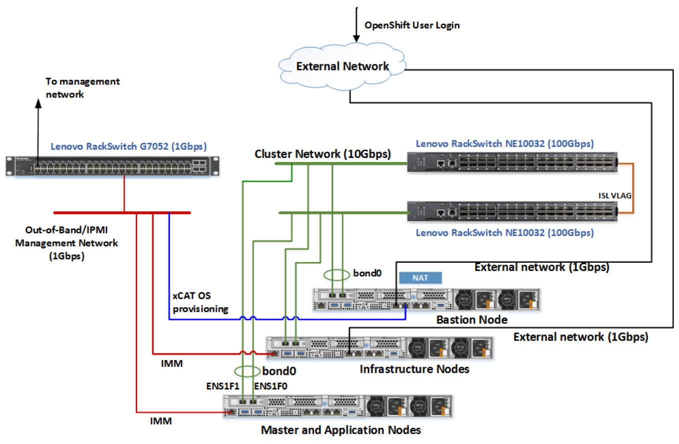


Figure 7. OpenShift logical/physical network connectivity

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.

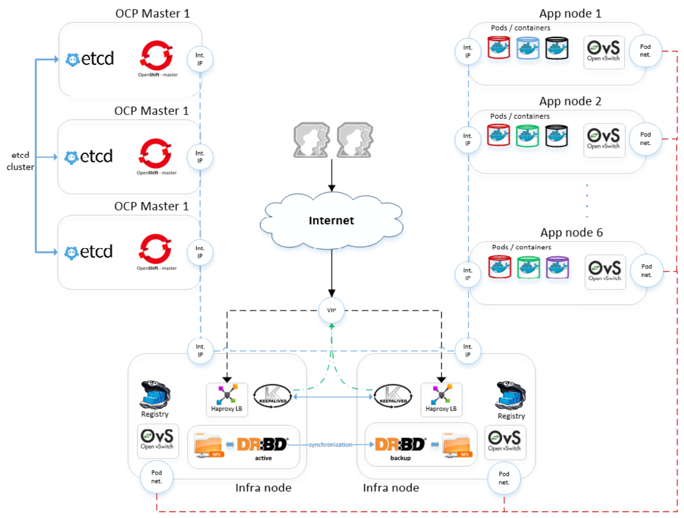


Figure 8. OpenShift Container Platform logical environment

Network Addresses

Table 5 shows the network and subnet addresses for the various networks in the OpenShift cluster.

Table 5. Network addresses

Network	Network purpose	VLAN ID	Interface
172.20.3.0/24	IPMI Network		In-band interface
172.30.4.0/22	Internal Network		bond0
10.240.37.0/22	External Network	546	eno1.546

Switch Management Addresses

Table 6 lists the management IP addresses for the switches.

Table 6. Switch addresses

Network	Network purpose	Interface
Lenovo ThinkSystem NE10032 RackSwitch	172.20.4.3	13
Lenovo ThinkSystem NE10032 RackSwitch	172.20.4.4	14
Lenovo RackSwitch G7052	172.20.4.1	

Server Addresses

Table 7. Server addresses

Host name	bond0	eno1.546	In-bound IMM
master1	172.30.4.9		172.20.3.9
master2	172.30.4.10		172.20.3.10
master3	172.30.4.11		172.20.3.11
infra1	172.30.4.7	VIP 10.240.37.132 VIP 10.240.37.133	172.20.3.7
infra2	172.30.4.8	VIP 10.240.37.132 VIP 10.240.37.134	172.20.3.8
app1	172.30.4.1		172.20.3.1
app2	172.30.4.2		172.20.3.2
app3	172.30.4.3		172.20.3.3
app4	172.30.4.4		172.20.3.4
app5	172.30.4.5		172.20.3.5
app6	172.30.4.6		172.20.3.6
b01	172.30.4.12	10.240.37.131	172.20.3.12

Table 9. Network connectivity table

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

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 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.

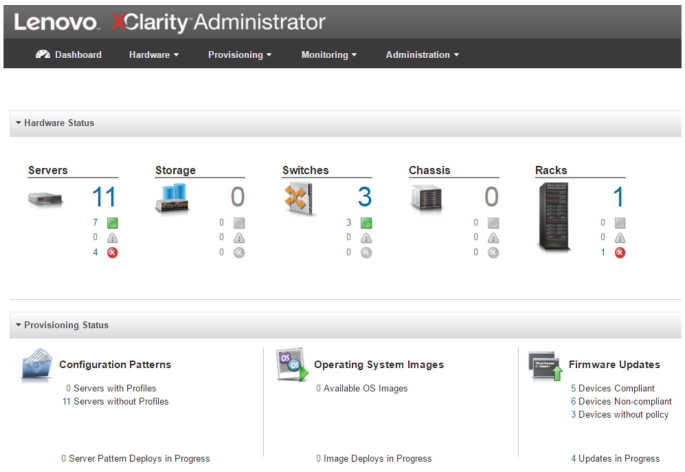
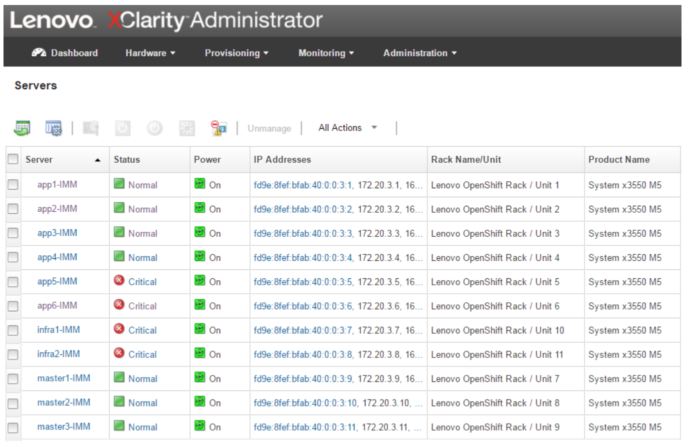


Figure 9. Lenovo XClarity Administrator graphical interface

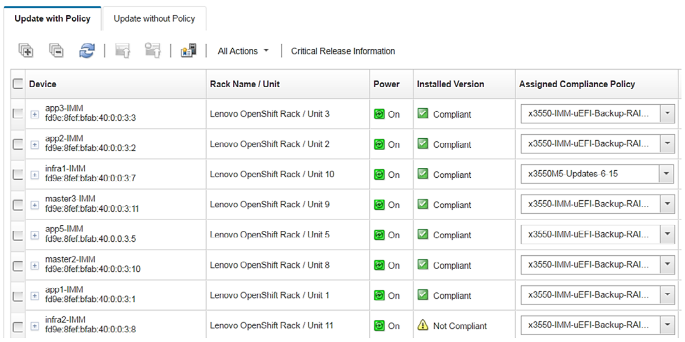
Lenovo XClarity Administrator features are summarized below.

- 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
- 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



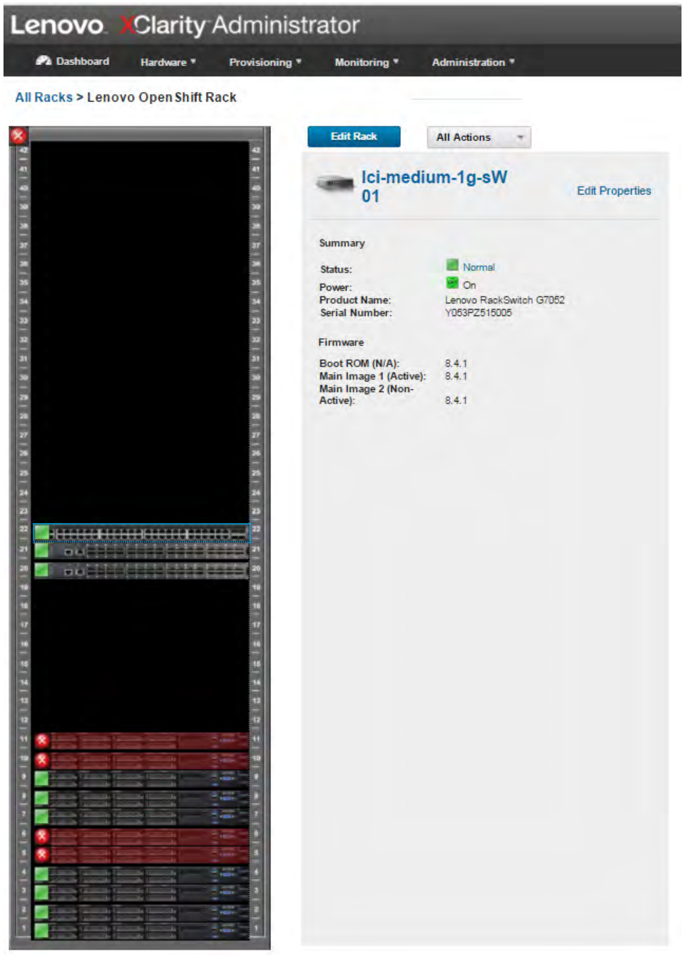
Server	Status	Power	IP Addresses	Rack Name/Unit	Product Name
app1-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:1, 172.20.3.1, 16...	Lenovo OpenShift Rack / Unit 1	System x3550 M5
app2-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:2, 172.20.3.2, 16...	Lenovo OpenShift Rack / Unit 2	System x3550 M5
app3-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:3, 172.20.3.3, 16...	Lenovo OpenShift Rack / Unit 3	System x3550 M5
app4-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:4, 172.20.3.4, 16...	Lenovo OpenShift Rack / Unit 4	System x3550 M5
app5-IMM	Critical	On	fd9e:8fef:bfab:40:0:0:3:5, 172.20.3.5, 16...	Lenovo OpenShift Rack / Unit 5	System x3550 M5
app6-IMM	Critical	On	fd9e:8fef:bfab:40:0:0:3:6, 172.20.3.6, 16...	Lenovo OpenShift Rack / Unit 6	System x3550 M5
infra1-IMM	Critical	On	fd9e:8fef:bfab:40:0:0:3:7, 172.20.3.7, 16...	Lenovo OpenShift Rack / Unit 10	System x3550 M5
infra2-IMM	Critical	On	fd9e:8fef:bfab:40:0:0:3:8, 172.20.3.8, 16...	Lenovo OpenShift Rack / Unit 11	System x3550 M5
master1-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:9, 172.20.3.9, 16...	Lenovo OpenShift Rack / Unit 7	System x3550 M5
master2-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:10, 172.20.3.10, ...	Lenovo OpenShift Rack / Unit 8	System x3550 M5
master3-IMM	Normal	On	fd9e:8fef:bfab:40:0:0:3:11, 172.20.3.11, ...	Lenovo OpenShift Rack / Unit 9	System x3550 M5

Figure 10. Server overview in Lenovo XClarity Administrator



Device	Rack Name / Unit	Power	Installed Version	Assigned Compliance Policy
app3-IMM fd9c:8fcf:bfab:40:0:0:3:3	Lenovo OpenShift Rack / Unit 3	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
app2-IMM fd9e:8fef:bfab:40:0:0:3:2	Lenovo OpenShift Rack / Unit 2	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
infra1-IMM fd9e:8fef:bfab:40:0:0:3:7	Lenovo OpenShift Rack / Unit 10	On	Compliant	x3550M5 Updates 6 15
master3-IMM fd9e:8fef:bfab:40:0:0:3:11	Lenovo OpenShift Rack / Unit 9	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
app5-IMM fd9e:8fef:bfab:40:0:0:3:5	Lenovo OpenShift Rack / Unit 5	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
master2-IMM fd9e:8fef:bfab:40:0:0:3:10	Lenovo OpenShift Rack / Unit 8	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
app1-IMM fd9e:8fef:bfab:40:0:0:3:1	Lenovo OpenShift Rack / Unit 1	On	Compliant	x3550-IMM-uEFI-Backup-RAI...
infra2-IMM fd9e:8fef:bfab:40:0:0:3:8	Lenovo OpenShift Rack / Unit 11	On	Not Compliant	x3550-IMM-uEFI-Backup-RAI...

Figure 11. Policy deployment controls in Lenovo XClarity Administrator



lci-medium-1g-sW 01

Summary

Status:	🟩 Normal
Power:	🟩 On
Product Name:	Lenovo RackSwitch G7052
Serial Number:	Y053PZ515005

Firmware

Boot ROM (N/A):	8.4.1
Main Image 1 (Active):	8.4.1
Main Image 2 (Non-Active):	8.4.1

Figure 12. Server management interface 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:

```
#!/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 number 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

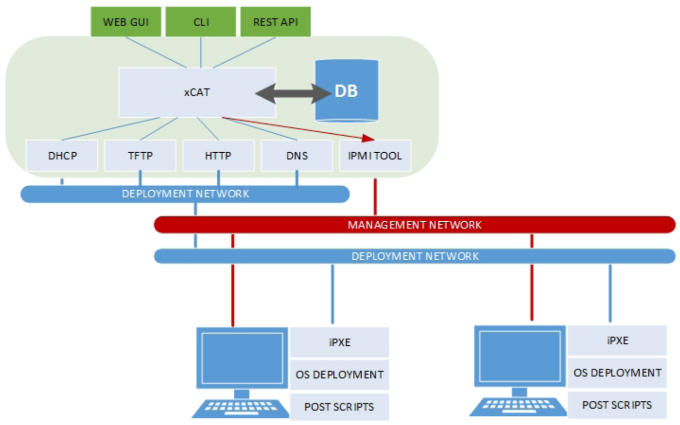


Figure 13. xCAT architecture

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:

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

Fill the main inventory file with switch information:

```
[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:

```
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:

```
$ 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, 1720.20.4.3 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 (rhels7/rhela7) node _
nic node _ type (app/infra/master)
1 6 rhela7 1 app
7 8 rhels7 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 -v /sys/fs/cgroup:/sys/fs/cgroup
--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.

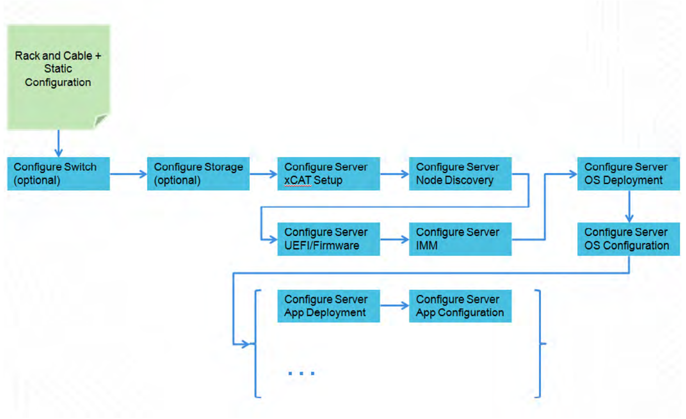
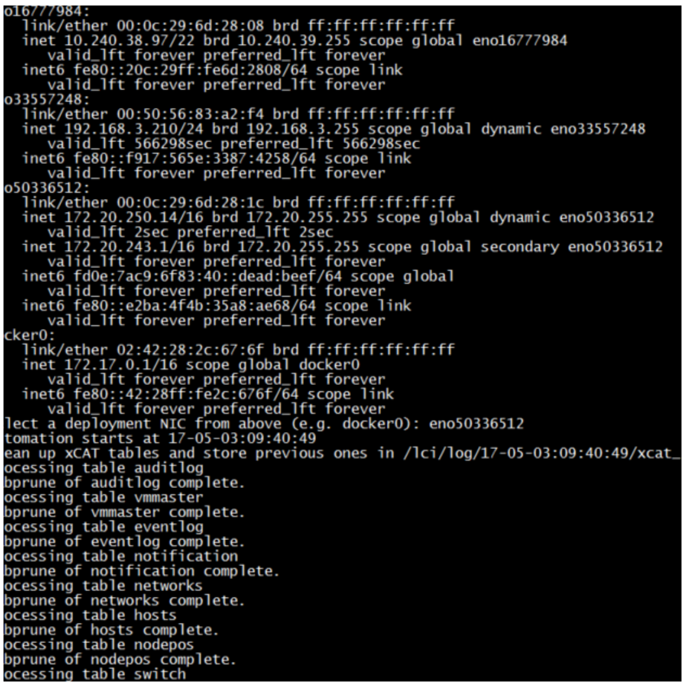

Rack and Cable + Static Configuration
Configure Switch (optional)
Configure Storage (optional)
Configure Server xCAT Setup
Configure Server Node Discovery
Configure Server OS Deployment
Configure Server UEFI/Firmware
Configure Server IMM
Configure Server OS Configuration
Configure Server App Deployment
Configure Server App Configuration
...

Figure 14. 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:



```
o16777984:
  link/ether 00:0c:29:6d:28:08 brd ff:ff:ff:ff:ff:ff
  inet 10.240.38.97/22 brd 10.240.39.255 scope global eno16777984
     valid_lft forever preferred_lft forever
  inet6 fe80::20c:29ff:fe6d:2808/64 scope link
     valid_lft forever preferred_lft forever
o33557248:
  link/ether 00:50:56:83:a2:f4 brd ff:ff:ff:ff:ff:ff
  inet 192.168.3.210/24 brd 192.168.3.255 scope global dynamic eno33557248
     valid_lft 566298sec preferred_lft 566298sec
  inet6 fe80::f917:565e:3387:4258/64 scope link
     valid_lft forever preferred_lft forever
o50336512:
  link/ether 00:0c:29:6d:28:1c brd ff:ff:ff:ff:ff:ff
  inet 172.20.250.14/16 brd 172.20.255.255 scope global dynamic eno50336512
     valid_lft 2sec preferred_lft 2sec
  inet 172.20.243.1/16 brd 172.20.255.255 scope global secondary eno50336512
     valid_lft forever preferred_lft forever
  inet6 fd0e:7ac9:6f83:40::dead:beef/64 scope global
     valid_lft forever preferred_lft forever
  inet6 fe80::e2ba:4f4b:35a8:ae68/64 scope link
     valid_lft forever preferred_lft forever
cker0:
  link/ether 02:42:28:2c:67:6f brd ff:ff:ff:ff:ff:ff
  inet 172.17.0.1/16 scope global docker0
     valid_lft forever preferred_lft forever
  inet6 fe80::42:28ff:fe2c:676f/64 scope link
     valid_lft forever preferred_lft forever
lect a deployment NIC from above (e.g. docker0): eno50336512
tomation starts at 17-05-03:09:40:49
ean up xCAT tables and store previous ones in /lci/log/17-05-03:09:40:49/xcat_
ocessing table auditlog
bprune of auditlog complete.
ocessing table vmmaster
bprune of vmmaster complete.
ocessing table eventlog
bprune of eventlog complete.
ocessing table notification
bprune of notification complete.
ocessing table networks
bprune of networks complete.
ocessing table hosts
bprune of hosts complete.
ocessing table nodepos
bprune of nodepos complete.
ocessing table switch
```

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]
infra1-mgmt: MAC Address 1: 08:94:ef:25:b1:93
infra1-mgmt: MAC Address 2: 08:94:ef:25:b1:94
infra1-mgmt: MAC Address 3: 08:94:ef:25:b1:95
infra1-mgmt: MAC Address 4: 08:94:ef:25:b1:96
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 1: 08:94:ef:25:a5:83
app1-mgmt: MAC Address 2: 08:94:ef:25:a5:84
app1-mgmt: MAC Address 3: 08:94:ef:25:a5:85
app1-mgmt: MAC Address 4: 08:94:ef:25:a5:86
master3-mgmt: MAC Address 1: 08:94:ef:25:a9:13
master3-mgmt: MAC Address 2: 08:94:ef:25:a9:14
master3-mgmt: MAC Address 3: 08:94:ef:25:a9:15
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)

```
master2-mgmt: MAC Address 2: 08:94:ef:25:a2:8c
master2-mgmt: MAC Address 3: 08:94:ef:25:a2:8d
master2-mgmt: MAC Address 4: 08:94:ef:25:a2:8e
infra2-mgmt: MAC Address 1: 08:94:ef:25:a2:3b
infra2-mgmt: MAC Address 2: 08:94:ef:25:a2:3c
infra2-mgmt: MAC Address 3: 08:94:ef:25:a2:3d
infra2-mgmt: MAC Address 4: 08:94:ef:25:a2:3e
app4-mgmt: MAC Address 1: 08:94:ef:25:a9:ab
app4-mgmt: MAC Address 2: 08:94:ef:25:a9:ac
app4-mgmt: MAC Address 3: 08:94:ef:25:a9:ad
app4-mgmt: MAC Address 4: 08:94:ef:25:a9:ae
app3-mgmt: MAC Address 1: 08:94:ef:25:af:53
app3-mgmt: MAC Address 2: 08:94:ef:25:af:54
app3-mgmt: MAC Address 3: 08:94:ef:25:af:55
app3-mgmt: MAC Address 4: 08:94:ef:25:af:56
IMM info:
app1-mgmt: 08:94:ef:25:a5:88 ()
app2-mgmt: 08:94:ef:25:aa:e0 ()
[3/1993]
app3-mgmt: 08:94:ef:25:af:58 ()
app4-mgmt: 08:94:ef:25:a9:b0 ()
app5-mgmt: 08:94:ef:25:b8:30 ()
app6-mgmt: 08:94:ef:25:b2:c0 ()
infra1-mgmt: 08:94:ef:25:b1:98 ()
infra2-mgmt: 08:94:ef:25:a2:40 ()
master1-mgmt: 08:94:ef:25:a3:60 ()
master2-mgmt: 08:94:ef:25:a2:90 ()
master3-mgmt: 08:94:ef:25:a9:18 ()
/etc/hosts:
127.0.0.1 localhost
172.30.4.1 app1 app1.ocp.example.local
172.20.2.1 app1-mgmt app1-mgmt.ocp.example.local app1
172.30.4.2 app2 app2.ocp.example.local
172.20.2.2 app2-mgmt app2-mgmt.ocp.example.local app2
172.30.4.3 app3 app3.ocp.example.local
172.20.2.3 app3-mgmt app3-mgmt.ocp.example.local app3
172.30.4.4 app4 app4.ocp.example.local
172.20.2.4 app4-mgmt app4-mgmt.ocp.example.local app4
172.30.4.5 app5 app5.ocp.example.local
172.20.2.5 app5-mgmt app5-mgmt.ocp.example.local app5
172.30.4.6 app6 app6.ocp.example.local
172.20.2.6 app6-mgmt app6-mgmt.ocp.example.local app6
172.20.4.1 gswitch1 gswitch1.ocp.example.local
172.20.4.2 gswitch2 gswitch2.ocp.example.local
172.20.4.3 gswitch3 gswitch3.ocp.example.local
172.20.4.4 gswitch4 gswitch4.ocp.example.local
172.30.4.7 infra1 infra1.ocp.example.local
172.20.2.7 infra1-mgmt infra1-mgmt.ocp.example.local
infra1
172.30.4.8 infra2 infra2.ocp.example.local
172.20.2.8 infra2-mgmt infra2-mgmt.ocp.example.local
infra2
172.30.4.9 master1 master1.ocp.example.local
172.20.2.9 master1-mgmt master1-mgmt.ocp.example.local
master1
172.30.4.10 master2 master2.ocp.example.local
172.20.2.10 master2-mgmt master2-mgmt.ocp.example.local
master2
172.30.4.11 master3 master3.ocp.example.local
172.20.2.11 master3-mgmt master3-mgmt.ocp.example.local
master3
172.20.3.1 app1-IMM
172.20.3.2 app2-IMM
172.20.3.3 app3-IMM
172.20.3.4 app4-IMM
172.20.3.5 app5-IMM
172.20.3.6 app6-IMM
172.20.3.7 infra1-IMM
172.20.3.8 infra2-IMM
172.20.3.9 master1-IMM
172.20.3.10 master2-IMM
172.20.3.11 master3-IMM
```

PREREQUISITES

Inventory

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.

```
[OSEv3:children]
nodes
masters
nfs
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 *nfs* (for Docker Registry persistent storage), *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': '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.

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

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

```
[nodes]
master1.ocp.example.local containerized=True
openshift_schedulable=False openshift_ip=172.30.4.9
openshift_hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True
openshift_schedulable=False openshift_ip=172.30.4.10
openshift_hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True
openshift_schedulable=False openshift_ip=172.30.4.11
openshift_hostname=master3.ocp.example.local
```

(Continued)

```
infra1.ocp.example.local containerized=True openshift_
schedulable=True openshift_node_labels="{'region':
'infra'}" openshift_ip=172.30.4.7 openshift_
hostname=infra1.ocp.example.local
infra2.ocp.example.local containerized=True openshift_
schedulable=True openshift_node_labels="{'region':
'infra'}" openshift_ip=172.30.4.8 openshift_
hostname=infra2.ocp.example.local

app1.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.1 openshift_
hostname=app1.ocp.example.local
app2.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.2 openshift_
hostname=app2.ocp.example.local
app3.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.3 openshift_
hostname=app3.ocp.example.local
app4.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.4 openshift_
hostname=app4.ocp.example.local
app5.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.5 openshift_
hostname=app5.ocp.example.local
app6.ocp.example.local containerized=True openshift_
schedulable=True openshift_ip=172.30.4.6 openshift_
hostname=app6.ocp.example.local
```

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.

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

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 10. Additional variables

Variable	Description
`rhel _ subscription _ user:`	Name of the user who will be used for registration
`rhel _ subscription _ pass:`	Password of the user who will be used for registration
`ansible _ ssh _ user:`	Insert root or other user with root privileges
`ansible _ become:`	Set to True to run commands with sudo privileges
`local _ dns:`	Type a proper IP address for your bastion node that runs the DNS service

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:

```
$ 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.

```
$ sudo subscription-manager register --username=<user _
name> \
   --password=<password>
```

Available subscriptions can be listed with the following command:

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

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

```
$ sudo subscription-manager attach --pool=<pool _ id>
```

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

```
$ 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:

```
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.

```
#[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:

```
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
```

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.

```
$ sudo yum -y install wget git net-tools bind-utils
iptables-services bridge-utils bash-completion
$ 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:

```
$ 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:

```
$ sudo atomic host upgrade
```

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

```
$ 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:

```
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:

```
$ 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:

```
$ 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:

```
global_defs {
   router_id LVS_DEVEL
}
vrrp_script haproxy_check {
   script "killall -0 haproxy"
   interval 2
   weight 2
}
vrrp_instance OCP_EXT {
   interface eno1.546
   virtual_router_id 51
   priority 98
   state  MASTER
   virtual_ipaddress {
       10.240.37.132 dev eno1.546
   }
   track_script {
       haproxy_check
   }
   authentication {
      auth_type PASS
      auth_pass RANDOM1
   }
}
vrrp_instance OCP_INT {
   interface bond0
   virtual_router_id 91
   priority 98
   state  MASTER
   virtual_ipaddress {
       172.30.4.132 dev bond0
   }
   track_script {
       haproxy_check
   }
   authentication {
      auth_type PASS
      auth_pass RANDOM2
   }
}
```

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

```
global_defs {
   router_id LVS_DEVEL
}
vrrp_script haproxy_check {
   script "killall -0 haproxy"
   interval 2
   weight 2
}
vrrp_instance OCP_EXT {
   interface eno1.546
   virtual_router_id 51
   priority 98
   state  MASTER
```

(Continued)

```
   virtual_ipaddress {
       10.240.37.132 dev eno1.546
   }
   track_script {
       haproxy_check
   }
   authentication {
      auth_type PASS
      auth_pass RANDOM1
   }
}
vrrp_instance OCP_INT {
   interface bond0
   virtual_router_id 91
   priority 98
   state  MASTER
   virtual_ipaddress {
       172.30.4.132 dev bond0
   }
   track_script {
       haproxy_check
   }
   authentication {
      auth_type PASS
      auth_pass RANDOM2
   }
}
```

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] secrion** in the inventory file (refer to Appendix A). In order to install DRBD automatically, perform the following steps.

```
$ git clone https://github.com/intel/openshift-container-
architecture/
$ su openshift
$ cd ra-redhat-openshift/src/drbd
$ ansible-playbook drbd.yml
```

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:

```
$ sudo lvcreate --name drbd _ registry --size 300G vg00
```

Next, DRBD needs to be installed on both nodes.

```
$ sudo rpm -ivh http://www.elrepo.org/elrepo-release-7.0-2.
el7.elrepo.noarch.rpm

$ 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 _ infra1-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:

```
$ 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*).

```
$ 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.

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

```
$ 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 onte of the 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
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 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
```

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.

```
$ oc get pods --namespace=default

NAME                        READY    STATUS    RESTARTS
AGE
docker-registry-2-qql92     1/1      Running   0
5m
docker-registry-2-uh7op     1/1      Running   0
5m
router-1-2vgcm              1/1      Running   0
5m
router-1-cbz87              1/1      Running   0         5m
```

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.

```
[OSEv3:children]
nodes
masters
nfs
etcd
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.

```
[new _ nodes]
app7.ocp.example.local containerized=True openshift _
schedulable=True
app8.ocp.example.local containerized=True openshift _
schedulable=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.

```
$ 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.

```
[nodes]
master1.ocp.example.local containerized=True openshift _
schedulable=False
master2.ocp.example.local containerized=True openshift _
schedulable=False
master3.ocp.example.local containerized=True openshift _
schedulable=False

infra1.ocp.example.local containerized=True openshift _
schedulable=True openshift _ node _ labels="{'region':
'infra'}"
infra2.ocp.example.local containerized=True openshift _
schedulable=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
app7.ocp.example.local containerized=True openshift _
schedulable=True
app8.ocp.example.local containerized=True openshift _
schedulable=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.

```
[OSEv3:children]
nodes
masters
nfs
etcd
lb
new _ masters
new _ nodes
```

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.

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

[new _ nodes]
master4.ocp.example.local containerized=True openshift _
schedulable=false
master5.ocp.example.local containerized=True openshift _
schedulable=false
```

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.

```
[masters]
master1.ocp.example.local containerized=True
master2.ocp.example.local containerized=True
master3.ocp.example.local containerized=True
master4.ocp.example.local containerized=True
master5.ocp.example.local containerized=True

[nodes]
master1.ocp.example.local containerized=True openshift _
schedulable=False
master2.ocp.example.local containerized=True openshift _
schedulable=False
master3.ocp.example.local containerized=True openshift _
schedulable=False
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 _
schedulable=True openshift _ node _ labels="{'region':
'infra'}"
infra2.ocp.example.local containerized=True openshift _
schedulable=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
```

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.

```
[OSEv3:children]
nodes
masters
nfs
etcd
lb
new _ etcd
```

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
etcd5.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
etcd2.ocp.example.local containerized=True
etcd3.ocp.example.local containerized=True
etcd4.ocp.example.local containerized=True
etcd5.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.

[1] Intel. "The Engine for Digital Transformation in the Data Center." March 2016. intel.com/content/www/us/en/processors/xeon/xeon-e5-brief.html.

Printed in USA 0717/CM/PRW/PDF Please Recycle 336171-001US

APPENDIX A: OCP DEPLOYMENT INVENTORY FILE

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

[OSEv3:vars]
ansible _ 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=eno1
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

rhel _ subscription _ user=RED _ HAT _ PORTAL _ LOGIN
rhel _ subscription _ pass=RED _ HAT _ PORTAL _ PASSWORD

external _ gateway=10.240.36.1

[local]
127.0.0.1

[masters]
master1.ocp.example.local containerized=True openshift _
ip=172.30.4.9 openshift _ hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift _
ip=172.30.4.10 openshift _ hostname=master2.ocp.example.
local
master3.ocp.example.local containerized=True openshift _
ip=172.30.4.11 openshift _ hostname=master3.ocp.example.
local
```

```
[nodes]
master1.ocp.example.local containerized=True openshift _
schedulable=False openshift _ ip=172.30.4.9 openshift _
hostname=master1.ocp.example.local
master2.ocp.example.local containerized=True openshift _
schedulable=False openshift _ ip=172.30.4.10 openshift _
hostname=master2.ocp.example.local
master3.ocp.example.local containerized=True openshift _
schedulable=False openshift _ ip=172.30.4.11 openshift _
hostname=master3.ocp.example.local

infra1.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.7 openshift _
hostname=infra1.ocp.example.local openshift _ node _
labels="{'region': 'infra'}"
infra2.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.8 openshift _
hostname=infra2.ocp.example.local openshift _ node _
labels="{'region': 'infra'}"

app1.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.1 openshift _
hostname=app1.ocp.example.local
app2.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.2 openshift _
hostname=app2.ocp.example.local
app3.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.3 openshift _
hostname=app3.ocp.example.local
app4.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.4 openshift _
hostname=app4.ocp.example.local
app5.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.5 openshift _
hostname=app5.ocp.example.local
app6.ocp.example.local containerized=True openshift _
schedulable=True openshift _ ip=172.30.4.6 openshift _
hostname=app6.ocp.example.local

[nfs]
nfs1.ocp.example.local openshift _ hostname=nfs1.ocp.
example.local openshift _ ip=172.30.4.7
nfs2.ocp.example.local openshift _ hostname=nfs2.ocp.
example.local openshift _ ip=172.30.4.8

[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

[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)