Reference Architecture: Red Hat OpenShift Container Platform on Lenovo ThinkSystem Servers

Last update: 01 October 2018
Version 1.0

Provides overview of application containers using Lenovo ThinkSystem Servers

Describes container orchestration technologies including Docker and Kubernetes

Describes DevOps and Continuous Integration and Continuous Delivery

Provides implementation example for OpenShift Container Platform

Srihari Angaluri

Xiaotong Jiang

Mike Perks

Billzheng Sun

Table of Contents

1 Introduction

The target audience for this Reference Architecture (RA) is system administrators or system architects. Some experience with Docker and OpenShift technologies may be helpful, but it is not required.

Emerging software applications are making use of containerization to enable rapid prototyping, testing, as well deployment to the cloud. The micro-service revolution introduced container-based virtualization, which offers many benefits when compared to traditional virtualization technologies. Containers provide a more portable and faster way to deploy services on cloud infrastructures compared to virtualization.

While containers themselves provide many benefits, they are not easily manageable in large environments. Hence, many container orchestration tools have increased in momentum and gained popularity. Each orchestration tool is different, hence they should be chosen individually for specific purposes. The Red Hat OpenShift® Container Platform uses Kubernetes which is an orchestration framework based on container-deployment practices. Kubernetes has gained popularity in the cloud community due to its maturity, scalability, performance, and many built-in tools that enable production-level container workload orchestration.

Red Hat OpenShift Container Platform 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. It provides many enterprise-ready features like enhanced security, multitenancy, simplified application deployment, and continuous integration/continuous deployment tools. With Lenovo™ servers and technologies, provisioning and managing the Red Hat OpenShift Container Platform infrastructure becomes effortless and produces a resilient solution.

This RA describes the system architecture for the Red Hat OpenShift Container Platform based on Lenovo ThinkSystem servers and Lenovo network switches. It 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 is provided for all required components to build the OpenShift cluster. An example deployment is used to show how to prepare, provision, deploy, and manage the Red Hat OpenShift Container Platform on Lenovo ThinkSystem servers and Lenovo network switches.

2 Business problem and business value

2.1 Business problem

Businesses today want to deliver new features and updates to their products for their internal users as well as external stakeholders quickly and with high quality. Every industry today is seeing a transformation, which is predominantly driven by advances in technology. In order to stay competitive and relevant in their respective industry and marketplace, every business needs to take advantage of new technologies quickly and adopt them to their products and solutions. Today, much of the technology advancement and innovation is driven through a combination of software and hardware. More importantly, emerging technologies such as artificial intelligence (AI) and machine learning (ML) are fuelled by rapid advancements in software. In addition, many of the IT infrastructure and data center advancements are driven through software defined technologies such as software defined storage (SDS) and software defined networking (SDN). Hence software is a key driver in pushing forward various technologies in all industries.

Container technology has picked up momentum in the software development area and enabled developers to take advantage of several benefits from packaging their applications as containers:

- Containers are light-weight application run-time environments compared to virtual machines and are therefore less resource intensive and highly efficient.
- Containers enable developers to package their applications as well as all the library dependencies to properly run them so that a container image provides a completely self-sufficient environment to execute the application code. This also means that multiple application instances requiring different versions of the same libraries can be packaged into different containers and run side-by-side on the same operating system instance without any interference.
- Containers are portable across different platforms (as long as the underlying operating systems are compatible). Docker is a well-known open source project that provides the run-time abstraction and facilities to build and run containerized applications in a portable fashion.
- Containers are now the de facto standard of operation for some of the well-known public cloud environments including Google, Amazon AWS, and Microsoft Azure. Hence, applications packaged as containers can be executed on-prem or on public cloud without any modifications (other than additional steps to security the software for use on the public cloud).
- There are many open source tools available now to help developers easily create, test, and deploy containerized applications. In addition, all the well-known protocols for security, authentication, /authorization, storage, etc., can be applied to containerized workloads without any modifications to applications. In other words, you can take a legacy application written in a language such as Java and package it, as is, into a container image and run it.
- Containers are now the way to implement a continuous integration/continuous delivery (CI/CD) development pipeline and the DevOps paradigm of combining software development and infrastructure operations.

2.2 Business value

Software development life cycle (SDLC) practices have evolved to achieve high velocity and efficiency of development. Organizations today implement Agile/Scrum as the primary methodology to create cohesive development teams that work close to their customers, gather incremental product requirements, and deliver new features in short development cycles.

2.2.1 DevOps Overview

DevOps is evolving as a standard practice in many organizations to bring together software development and IT operations teams for the goal of eliminating process bottlenecks in development, quality assurance (QA), and delivery cycles, and servicing their end customers efficiently. Implementing a proper DevOps process requires careful planning and an assessment of the end-to-end pipeline from development to QA to delivery. Automation is a key aspect of DevOps. Traditional software development processes were handled mostly manually. When developers commit code to the source code repository, the test engineer or a build engineer would then checkout the code and build the project, resolving any conflicts. After the QA iterations, the release engineer would be responsible to take the release branch code and build the final shippable product. Along this pipeline, many of the steps were handled manually by people, which introduced the delays in the release cycles. Agile development now takes advantage of new automation tools that remove the manual steps.

Another core aspect of DevOps is providing the necessary freedom and resources to develop and test code without having to rely on IT operations teams to re-provision or re-configure hardware every time. With the advances in technologies including virtualization, containers, Cloud multi-tenancy, self-service, and so on, it is now possible to detach applications and end-users from physical hardware and provide the necessary tools for them to create the right virtual environment to run their applications without directly modifying the physical hardware or interfering with other users' applications. With cloud self-service, users can request and provision the hardware to meet their application specific requirements. Cloud administrators create the proper policy and authorization workflows such that the provisioning process does not require manual steps. DevOps essentially combines the role of the software engineer with that of the IT operator so that the end-to-end software pipeline can be implemented with automation.

2.2.2 Monolithic Vs Micro-services Architecture

Software architecture over the last two to three decades has evolved from a monolithic application that essentially delivered all feature functions in a single package to service-oriented architectures where the application is divided up into multiple tiers with each tier providing programming interfaces (APIs) for its clients to access the features via service calls. Software principles such as modularity, coupling, code reuse, etc., have remained the core principles that people still use, however new programming languages, runtime facilities, mobile versus cloud native techniques, etc., have evolved in the recent years to shift software from the traditional architectures to more of a micro-service architecture.

Micro-services are software applications that are organized around smaller subsets of functionality of the overall application such that they are much more manageable than a bulky piece of software developed by 10s of developers and coordinated in a complex dev/test process. Micro-services are software modules that run as services with open APIs. They use open protocols, e.g. HTTP, and expose REST based APIs so that the services can run anywhere - on-premises or on the public cloud, and still be locatable via well-defined service end-points. Due to this design, micro-services provide a loosely-coupled architecture that can be maintained by smaller development teams and can be independently updated.

Containers provide a natural mechanism to implement micro-services because they allow you to package the application code and all its runtime library dependencies into a single image, which is portable across various platforms. In addition, container orchestration platforms such as Kubernetes provide the mechanisms for service location, routing, service replication, etc., which helps micro-services development and runtime. Developers do not need to explicitly write additional code for these types of services because the platform provides these facilities.

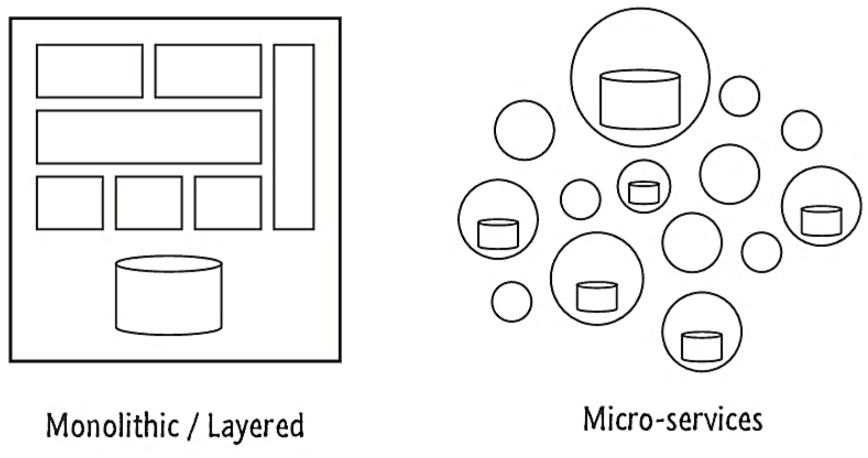


Figure 1. Moving from a monolithic to Micro-services architecture

2.2.3 Continuous Integration/Continuous Delivery (CI/CD)

As discussed in the previous section, successful DevOps practice requires a good amount of automation of the development, test, QA, and delivery pipeline. This is where the CI/CD comes into play.

Continuous integration is the process by which new code development through build, unit testing, QA, and delivery is automated end-to-end using build tools and process workflows. CI enables rapid integration of code being developed by multiple engineers concurrently and committed into a source code repository. CI enables rapid build and test of code so that software bugs and quality issues are identified quickly. Once the code passes the test plan and QA it can then be pushed to the release branches for release integration.

Continuous Delivery (CD) enables automation around delivering code to production systems after performing the necessary functional, quality, security, and performance tests. Continuous delivery enables bringing new features in the software to the end users faster without going through the manual release test and promotion steps.

More information on CI/CD with OpenShift is available in the following online book:
assets.openshift.com/hubfs/pdfs/DevOps_with_OpenShift.pdf

Figure 2 shows a high-level view of DevOps pipeline.

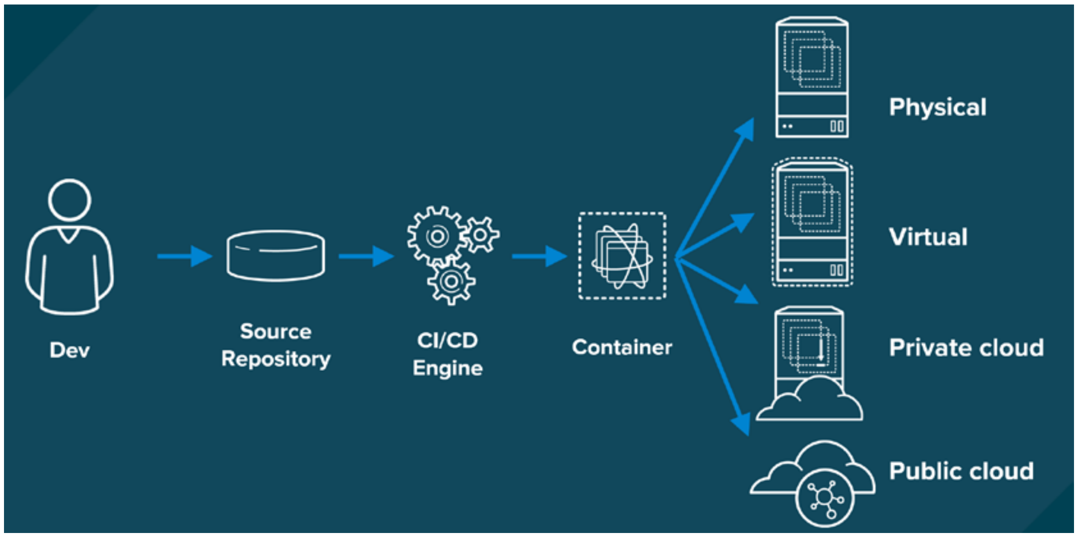


Figure 2. DevOps pipeline from a high-level

As described previously, source code from multiple concurrent developers is integrated, tested, and deployed to production through automation tools. The OpenShift Container Platform provides the mechanisms to implement the CI/CD pipelines with tools such as CloudBees Jenkins. See the following post on how to do this: blog.openshift.com/cicd-with-openshift/.

3 Requirements

The functional and non-functional requirements for this reference architecture are desribed below.

3.1 Functional requirements

Table 1 lists the functional requirements.

Table 1. Functional Requirements

Requirement	Description
Container orchestration services	Red Hat OpenShift Container Platform is designed to run workload container images at scale using the Kubernetes container orchestrator, Docker container runtime, and container runtime interface (CRI-O).
User self-service	OpenShift supports a Web based UI console that allows users to login and manage their containerized workloads.
Policy management	OpenShift allows administrators to configure role-based authorization to manage the system resources such as compute, networking, and storage, and application workloads.
Cloud integration	OpenShift supports an integrated container registry, the Quay container registry, or public registries which allow users to pull down container images from other places. In addition, building container images on OpenShift platform allows portability to other clouds such as Google container engine.
Network and Storage virtualization	Through built-in OpenShift networking and storage services for Kubernetes, users can access these abstracted resources through their container applications. In addition, OpenShift provides network infrastructure services through open protocols such as VXLAN. A variety of storage facilities can be exposed to container applications via the Kubernetes persistent volume plug-ins and stateful sets.
Command line tools	OpenShift container platform provides CLI tools for almost all cluster operations and for container image operations. In addition, administrators can use Kubernetes CLI tools to directly access its services.
CI/CD tools	A variety of open source and commercial tools are available such as Jenkins build server and integration with GitHub source code repository to implement CI/CD pipelines.
Open source tools	Red Hat container registry and other open container registries such as dockerhub are available to OpenShift users to access open source tools such as nginx, apache httpd, mysql, postgres, cassandra, etc.
Automation tools	Many tools are available for automation including Ansible, Chef, Puppet, etc.

3.2 Non-functional requirements

Table 2 lists the non-functional requirements that are needed for typical OpenShift deployments

Table 2. Non-functional Requirements

Requirement	Description
Scalability	The OpenShift Container Platform is designed for scale. The platform allows for hundreds of containerized workloads to be scheduled and run without any performance bottlenecks. The physical resources such as compute nodes and storage can be scaled as the workload and user base grows.
Load balancing	OpenShift master nodes provide the core API and management services for the Kubernetes cluster. For production environments, you can implement load-balancing and heartbeat monitoring for the core services via HAproxy and Keepalived. In addition, Kubernetes handles load-balancing of the workload containers through built-in scheduler features, network routing, replication services, etc.
Fault tolerance	Fault tolerance can be provided to critical container workloads such as databases via Kubernetes built-in mechanisms. In addition, data and configuration settings for container images can be persisted across instances via persistent volume claims and stateful sets.
Physical footprint	OpenShift container platform can be implemented with as little as a single node where all services are consolidated and scaled through multiple physical nodes to distribute services and containers. Hence, the architecture is quite flexible and allows to start small and then scale out.
Ease of installation	Ansible playbooks are available to automate the OpenShift Container Platform deployment.
Ease of management/operations	Administrator tools and OpenShift Web console allow day 2 management operations to be performed. In addition, the Lenovo XClarity Administrator tool enables hardware monitoring and management.
Flexibility	OpenShift container platform can be deployed both in a development/test setting and production setting. Various options are available for third party network and storage implementation for OpenShift.
Security	Red Hat OpenShift Container Platform has built-in enterprise grade security, all the way from the operating system layer up to the container registries. Both built-in authentication/authorization facilities and external authentication/authorization integration with tools such as OpenLDAP are supported.
High performance	OpenShift and Kubernetes have achieved wide industry adoption due to the robustness of the platform and high-performance. Enterprises can implement very large-scale OpenShift environments to support hundreds of users and thousands of container workloads with no performance bottlenecks.

4 Architectural overview

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 all of the architecture, processes, platforms, and services needed to help development and operations teams traverse traditional siloed structures and produce applications that help businesses succeed.

Figure 3 below shows the high level architecture of the Red Hat OpenShift Container Platform and the core building blocks. OpenShift is a platform designed to orchestrate containerized workloads across a cluster of nodes. The system uses the Kubernetes as the core container orchestration engine, which manages the Docker container images and their lifecycle.

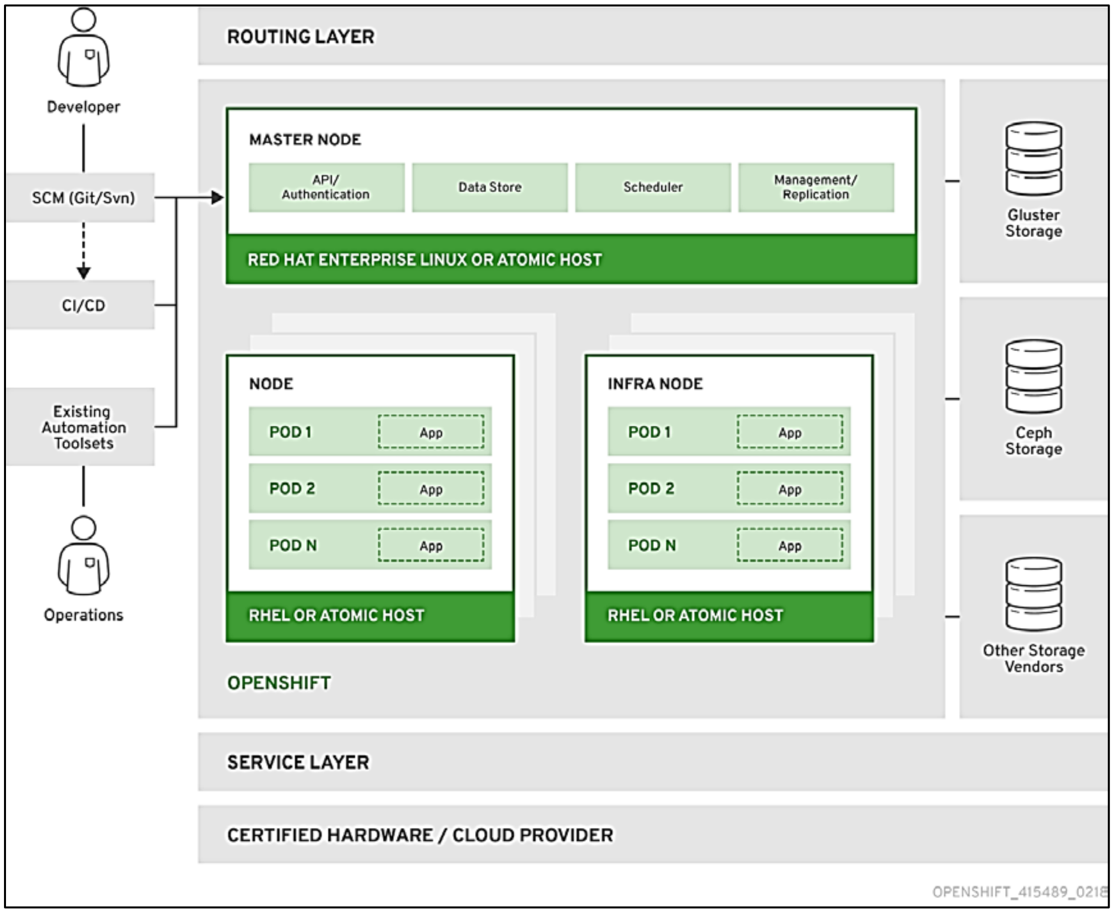


Figure 3. Red Hat OpenShift Container Platform Architecture

The physical configuration of the OpenShift platform is based on the Kubernetes cluster architecture. The master node is the primary node on which the Kubernetes scheduler, along with the distributed cluster data store (etcd), the REST API services, and other associated management services run. In a product environment, you need to ensure high availability of the master services through replicating the services to multiple physical servers and implementing monitoring and load-balancing services such as Keepalived and HAproxy. The infrastructure nodes can be used in a product setting to implement such services.

Application nodes (or just shown as Node in the diagram) run the users containerized applications on top of the Docker container environment.

5 Component model

As shown in Figure 4, this chapter describes the components and logical architecture of the Red Hat OpenShift solution.

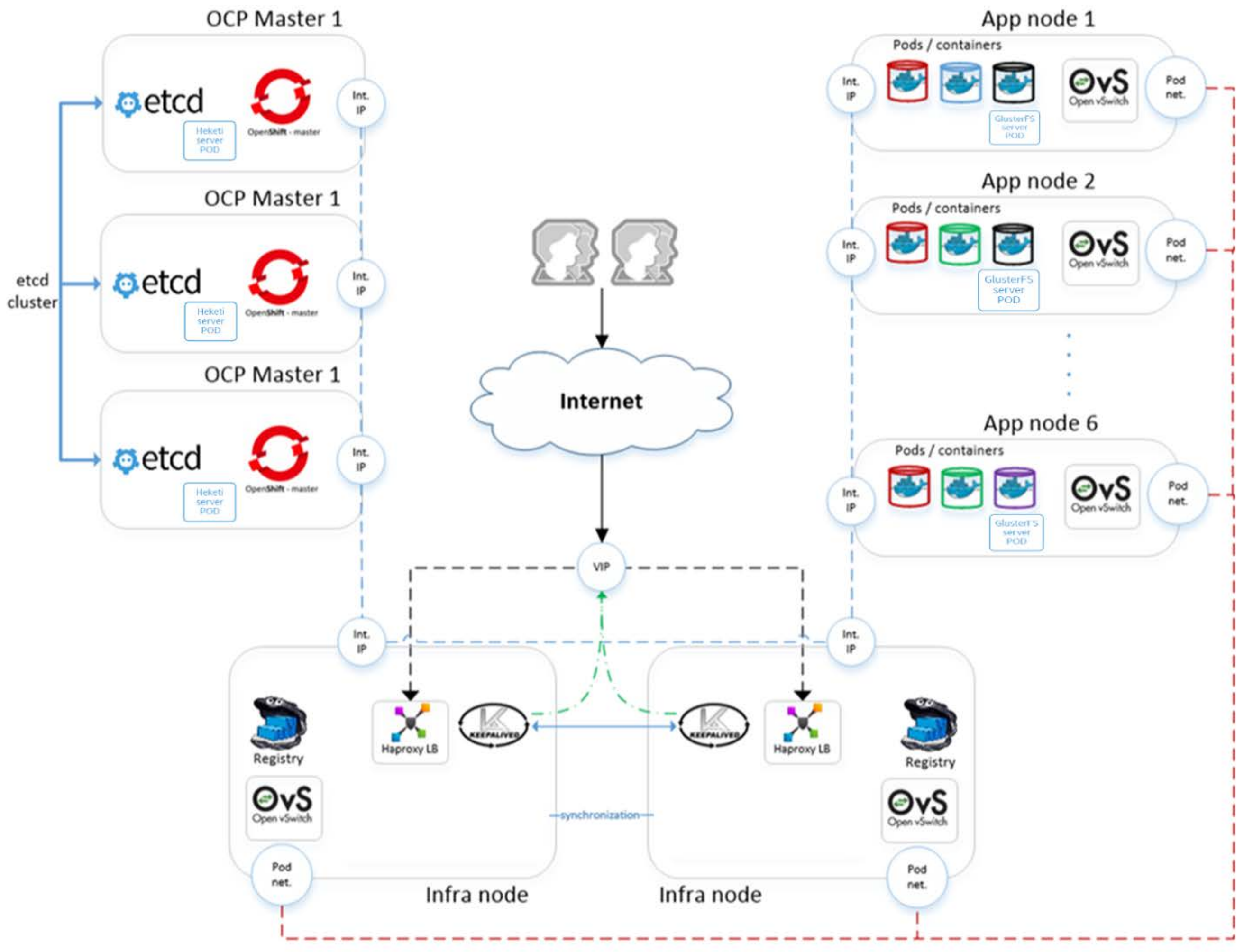


Figure 4. Red Hat OpenShift Container Platform logical architecture

All the 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 a Red Hat OpenShift Container Storage server that shares disk space with Docker Registry for Docker image storage. This storage is backed up by Red Hat OpenShift Container Storage, so Docker Registry storage can be easily switched in case of a node failure.

5.1 OpenShift infrastructure components

Figure 5 shows the four types of OpenShift nodes: *bastion*, *master*, *infrastructure,* and *application*.

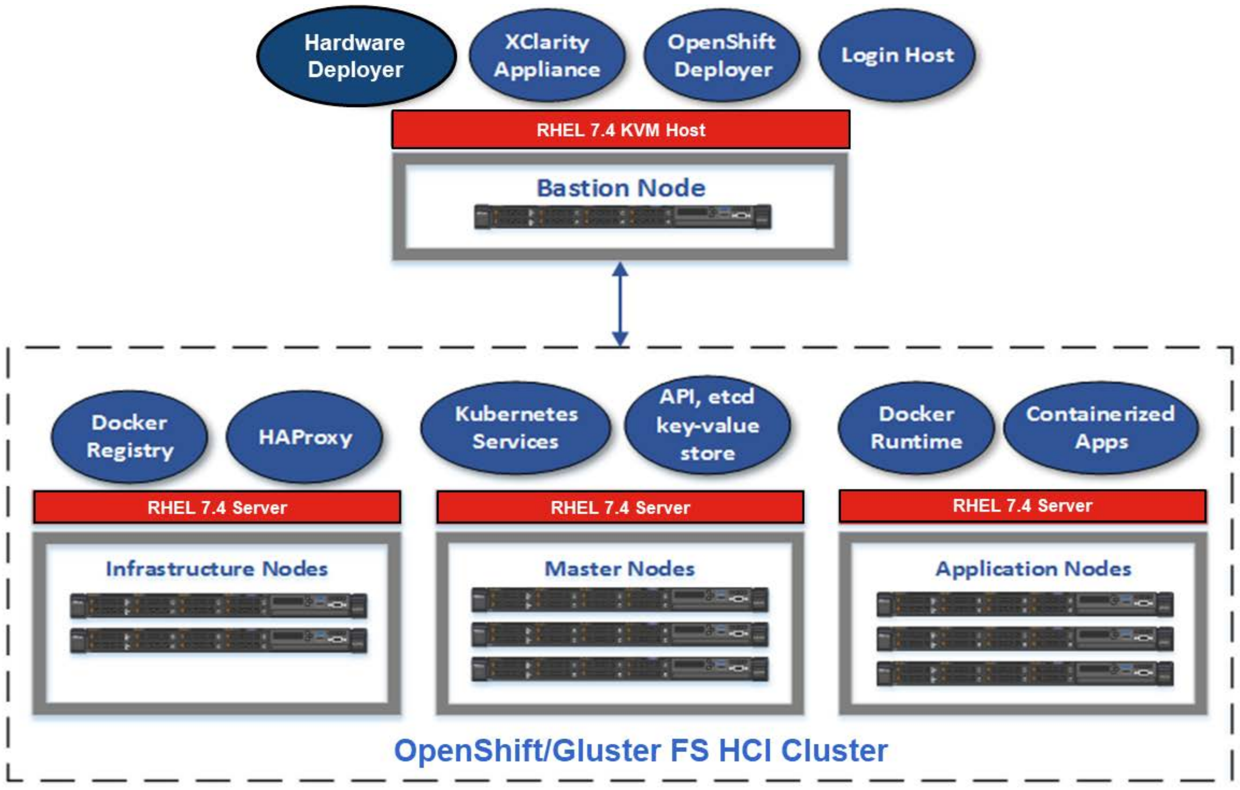


Figure 5. OpenShift Nodes

5.1.1 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. In addition, this node is also used for hardware management via tools such as xCAT and Lenovo XClarity Administrator. The *Bastion* node runs RHEL 7.4 Server with the Linux KVM packages installed.

5.1.2 Master node

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 either RHEL Atomic Host or RHEL 7.4 Server.

5.1.3 Infrastructure node

The OpenShift *infrastructure* node runs infrastructure-specific services such as the Docker Registry and the HAProxy router. The 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 Server 7.4.

5.1.4 Application node

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.

5.2 OpenShift architecture

Kubernetes is an open source project developed by Google. The project gained popularity via its open and flexible architecture for managing containerized workloads at large scale. It provides APIs that can be easily integrated into other tools such as the Red Hat OpenShift Container platform. Kubernetes provides the orchestration capabilities for containers, including scheduling the container images to nodes in a cluster, managing the container life cycle, availability, replication, persistent and non-persistent storage for containers, policy, multi-tenancy, network virtualization, routing, hierarchical clusters via federation APIs, and so forth.

A detailed software description of the Kubernetes components is described on this website: docs.openshift.com/enterprise/3.0/architecture/infrastructure_components/kubernetes_infrastructure.html.

Figure 6 shows the OpenShift high-level architecture and components.

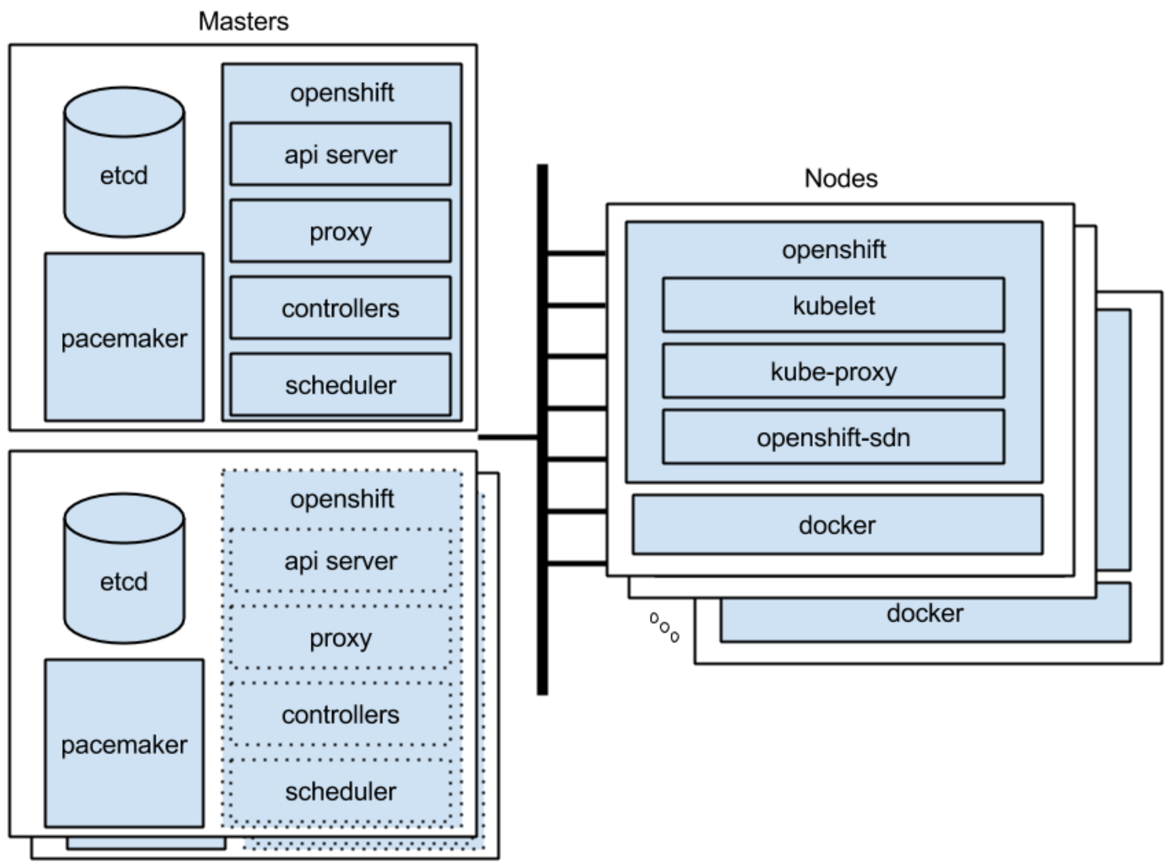


Figure 6. OpenShift component architecture

The master nodes, as described previously, are responsible for core services such as API interface, authentication/authorization, container scheduling, controller management, and configuration database. The master manages the state of the cluster and the lifecycle of the user container images. For redundancy and high availability, you can have multiple master nodes with frontend load-balancers such as HAproxy. The command line interface to the master nodes is implemented via the “oc” command.

The (worker) nodes are where users’ container images are executed. In OpenShift terminology the worker nodes run “pods”, each of which manages one or more running containers. Each node implements a “kubelet”, which is the node level controller that manages the pods and interacts with the OpenShift master.

In addition to the core OpenShift services, the Red Hat OpenShift platform also includes other features such as the Web based user self-service console, monitoring, an integrated container registry, storage management, authentication/authorization, automation via built-in Ansible playbooks, and other administrative tools for managing the container platform.

6 Operational model

This chapter describes the options for mapping the logical components of Red Hat OpenShift onto Lenovo ThinkSystem servers, storage, and Lenovo network switches.

6.1 Hardware components

The following section describes the hardware components that can be used in an OpenShift implementation.

6.1.1 Lenovo ThinkSystem D2 Chassis and SD530 Dense Server

The Lenovo ThinkSystem SD530 dense offering fits four hot-pluggable SD530 servers into a ThinkSystem D2 Enclosure. The enclosures each take up only 2U (0.5U per server).

- Each SD530 server supports two processors from the Intel Xeon processor Scalable family, up to 16 DIMMs, 6 drive bays, and two PCIe slots.
- Each SD530 server supports up to six 2.5-inch hot-swap drives. Two drive bays can be configured to support NVMe drives to maximize I/O performance in terms of throughput, bandwidth, and latency.
- The server has two optional 10Gb Ethernet ports, either 10GBASE-T or SFP+, routed from the embedded X722 controller to the optional 8-port EIOM module at the rear of the enclosure.
- The server includes an XClarity Controller (XCC) to monitor server availability.
- Lenovo XClarity Administrator offers comprehensive hardware management tools that help to increase uptime, reduce costs and improve productivity through advanced server management capabilities.
- New UEFI-based Lenovo XClarity Provisioning Manager, accessible from F1 during boot, provides system inventory information, graphical UEFI Setup, platform update function, RAID Setup wizard, operating system installation function, and diagnostic functions.

For more detailed product guide on the ThinkSystem D2 chassis and SD530 server, see: Lenovo ThinkSystem SD530 Server Product Guide.

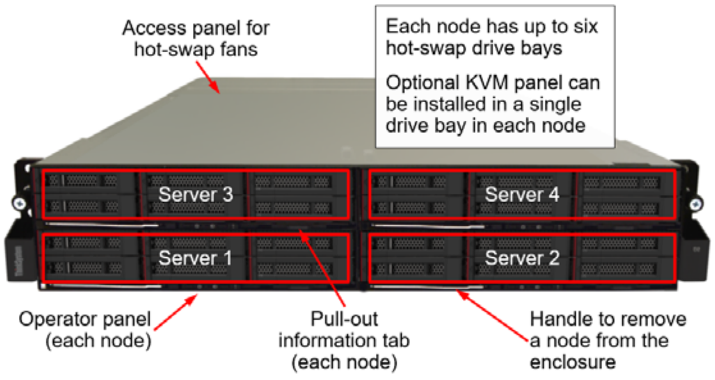

Access panel for hot-swap fans
Each node has up to six hot-swap drive bays
Optional KVM panel can be installed in a single drive bay in each node
Server 3
Server 4
Server 1
Server 2
Operator panel (each node)
Pull-out information tab (each node)
Handle to remove a node from the enclosure

Figure 7. Lenovo ThinkSystem D2 enclosure and SD530 server nodes

6.1.2 Lenovo ThinkSystem SR630 1U Server

Lenovo ThinkSystem SR630 is a 2-socket 1U rack server for enterprises that need industry-leading reliability, management, and security, as well as maximizing performance and flexibility for future growth. The SR630 server is designed to handle a wide range of workloads, such as databases, virtualization and cloud computing, virtual desktop infrastructure (VDI), infrastructure security, systems management, enterprise applications, collaboration/email, streaming media, web, and HPC.

- ThinkSystem SR630 supports two Intel Xeon Processor Scalable Family processors with up to 28-core processors, up to 38.5 MB of last level cache (LLC), up to 2666 MHz memory speeds, and up to 10.4 GT/s Ultra Path Interconnect (UPI) links.
- Offers flexible and scalable internal storage in a 1U rack form factor with up to 12x 2.5-inch drives for performance-optimized configurations or up to 4x 3.5-inch drives for capacity-optimized configurations, providing a wide selection of SAS/SATA HDD/SSD and PCIe NVMe SSD types and capacities.
- Provides I/O scalability with the LOM slot, PCIe 3.0 slot for an internal storage controller, and up to three PCI Express (PCIe) 3.0 I/O expansion slots in a 1U rack form factor.

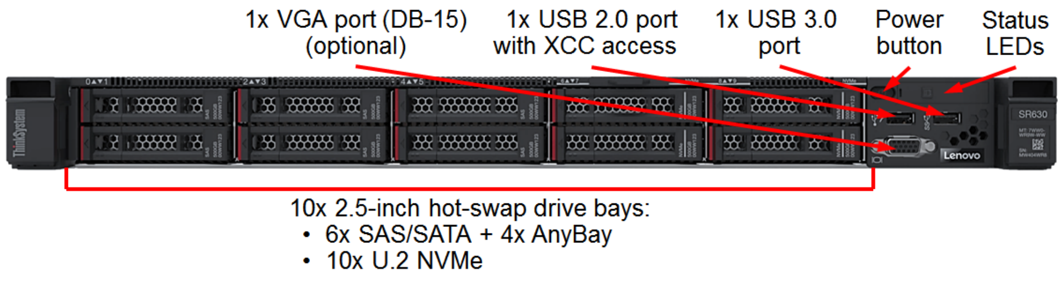

1x VGA port (DB-15) (optional)
1x USB 2.0 port with XCC access
1x USB 3.0 port
Power button
Status LEDs
10x 2.5-inch hot-swap drive bays:
• 6x SAS/SATA + 4x AnyBay
• 10x U.2 NVMe

Figure 8. Lenovo ThinkSystem SR630 Server front and rear views

More detailed information, see: ThinkSystem SR630 Server Product Guide.

6.1.3 Lenovo ThinkSystem NE1032/NE1032T Rack Switch

The Lenovo ThinkSystem NE1032/NE1032T RackSwitch family is a 1U rack-mount 10 Gb Ethernet switch that delivers lossless, low-latency performance with feature-rich design that supports virtualization, Converged Enhanced Ethernet (CEE), high availability, and enterprise class Layer 2 and Layer 3 functionality. The hot-swap redundant power supplies and fans (along with numerous high-availability features) help provide high availability for business sensitive traffic. These switches deliver line-rate, high-bandwidth switching, filtering, and traffic queuing without delaying data.

The NE1032 RackSwitch has 32x SFP+ ports that support 1 GbE and 10 GbE optical transceivers, active optical cables (AOCs), and direct attach copper (DAC) cables.

Figure 9 Lenovo ThinkSystem NE1032 RackSwitch

For more information, see the ThinkSystem NE1032 Product Guide.

The NE1032T RackSwitch has 24x 1/10 Gb Ethernet (RJ-45) fixed ports and 8x SFP+ ports that support 1 GbE and 10 GbE optical transceivers, active optical cables (AOCs), and direct attach copper (DAC) cables.

Figure 10 Lenovo ThinkSystem NE1032T RackSwitch

For more information, see the ThinkSystem NE1032T Product Guide.

6.1.4 Lenovo ThinkSystem NE2572 RackSwitch

For scale-out OpenShift Container Platform implementations to support 1000s of container images with high-performance requirements for network as well as storage, it is recommended to use 25Gbps Ethernet networking as the fabric.

The Lenovo ThinkSystem NE2572 RackSwitch is designed for the data center and provides 10 Gb/25 Gb Ethernet connectivity with 40 Gb/100 Gb Ethernet upstream links. It is ideal for big data, cloud, and enterprise workload solutions. It is an enterprise class Layer 2 and Layer 3 full featured switch that delivers line-rate, high-bandwidth switching, filtering, and traffic queuing without delaying data.

The NE2572 RackSwitch has 48x SFP28/SFP+ ports that support 10 GbE SFP+ and 25 GbE SFP28 optical transceivers, active optical cables (AOCs), and direct attach copper (DAC) cables. The switch also offers 6x QSFP28/QSFP+ ports that support 40 GbE QSFP+ and 100 GbE QSFP28 optical transceivers, active optical cables (AOCs), and direct attach copper (DAC) cables. The QSFP28/QSFP+ ports can also be split out into two 50 GbE (for 100 GbE QSFP28), or four 10 GbE (for 40 GbE QSFP+) or 25 GbE (for 100 GbE QSFP28) connections by using breakout cables.

Figure 11. Lenovo ThinkSystem NE2572 Rack Switch

For more information, see the Lenovo ThinkSystem NE2572 Switch Product Guide.

6.1.5 Lenovo RackSwitch G8052

The Lenovo System Networking RackSwitch G8052 (as shown in Figure 12) is an Ethernet switch that is designed for the data center and provides a virtualized, cooler, and simpler network solution. The Lenovo RackSwitch G8052 offers up to 48 1 GbE ports and up to 4 10 GbE ports in a 1U footprint. The G8052 switch is always available for business-sensitive traffic by using redundant power supplies, fans, and numerous high-availability features. For more information, see this website: lenovopress.com/tips1270.

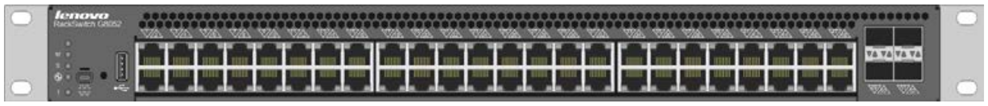


Figure 12: Lenovo RackSwitch G8052

6.2 Deployment models

The OpenShift Container Platform can be implemented in development/test, staging, and production settings. Each node role has its own dedicated servers for performance and availability. However, in a non-production environment, a minimal environment can be provided to test applications before moving them to a staging or production environment.

For a test/development environment, you can use minishift or ‘oc cluster up’ to deploy all on one server, or implement OpenShift with five servers for a more formal platform as shown in table below:

Node type	Quantity	Node role
Deployment	1	Deployment of the environment, Ansible playbooks, hardware management, etc.
Master	1	OpenShift API master, Kubernetes scheduler, etcd, other core services
Compute	3	Runs the application containers

For a production OpenShift deployment, all of the core services such as the API servers, Kubernetes scheduler, etcd, etc., need to be highly available. The table below shows the recommended configuration for a production deployment.

Node type	**Quantity**	**Node role**
Bastion	1	Deployment of the environment, Ansible playbooks, hardware management, etc.
Infrastructure	2	OpenShift HAProxy, container registry, routing, etcd, logging, metrics.
Master	3	OpenShift API master, Kubernetes scheduler
Compute	3+	Runs the application containers
Compute with Red Hat OpenShift Container Storage	3+	Compute nodes that run Gluster FS container to provide hyperconverged compute and storage

There are performance and availability implications of running the Red Hat OpenShift Container Storage alongside the workload containers in a hyperconverged environment. For production environments, it is recommended to separate hyper-converged compute servers from storage-only servers, or to ensure that the servers have sufficient CPU, memory, and storage resources to avoid any performance bottlenecks.

6.3 Compute servers

The OpenShift Container Platform can be implemented on a small footprint of x86 servers clustered together and scaled as the user workloads grow.

The right choice of servers and the corresponding hardware configuration for CPUs, memory, and networking will depend upon various factors, including but not limited to:

- Number of concurrent OpenShift users to be supported
- Type and mix of application workloads, which will drive the system resource requirements
- System growth projection
- Development or production use
- Fault-tolerance and availability requirements for applications
- Application performance expectations
- Implementation of hybrid-cloud model, which drives the requirements for on-premises infrastructure

For more guidance on sizing and other considerations is available for OpenShift clusters in OpenShift documentation: access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/scaling_and_performance_guide/#scaling-performance-cluster-limits

Lenovo does not recommend server configuration specifics for CPU, memory, storage, etc. because it is heavily dependent on the sizing considerations previously listed. The user is requested to perform a proper sizing assessment for their particular needs and choose the hardware configurations to meet those requirements.

6.4 Persistent storage for containerized workloads

There are two types of storage consumed by containerized applications – ephemeral (non-persistent) and persistent. As the names suggest, non-persistent storage is created and destroyed along with the container and is only used by applications during their lifetime as a container. Hence, non-persistent storage is used for temporary data. When implementing the OpenShift Container Platform, local disk space on the application nodes can be configured and used for the non-persistent storage volumes.

Persistent storage, on the other hand, is used for data that needs to be persisted across container instantiations. An example is a 2 or 3-tier application that has separate containers for the web and business logic tier and the database tier. The web and business logic tier can be scaled out using multiple containers for high availability. The database that is used in the database tier requires persistent storage that is not destroyed.

OpenShift uses a persistent volume framework that operates on two concepts – persistent storage and persistent volume claim. Persistent storage is the physical storage volumes that are created and managed by the OpenShift cluster administrator. When an application container requires persistent storage, it would create a persistent volume claim (PVC). The PVC is a unique pointer/handle to a persistent volume on the physical storage, except that PVC is not bound to a physical volume. When a container makes a PVC request, OpenShift would allocate the physical disk and binds it to the PVC. When the container image is destroyed, the volume bound to the PVC is not destroyed unless you explicitly destroy that volume. In addition, during the lifecycle of the container if it relocates to another physical server in the cluster, the PVC binding will still be maintained. After the container image is destroyed, the PVC is released, but the persisted storage volume is not deleted. The specific persistent storage policy for the volume will determine when the volume gets deleted.

For more detailed conceptual information on persistent volumes see: access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/architecture/#architecture-additional-concepts-storage

A variety of persistent storage options are available for OpenShift, choices including NFS, OpenStack Cinder, Ceph RBD, iSCSI, fiber channel SAN, hyperconverged storage using Red Hat OpenShift Container Storage, AWS elastic block storage (EBS), and others. For a complete list of these choices and the corresponding requirements, see the link below: access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#configuring-persistent-storage

6.4.1 Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage is used as the persistent storage backend in this Reference Architecture as it simplifies the overall OpenShift architecture and consolidates the compute and storage components in the same x86 servers.

Red Hat OpenShift Container Storage is based on Gluster which is an open source distributed, scalable, and high-performance file based storage system. It is used widely for many types of applications. Red Hat OpenShift Container Storage provides volume plug-ins into OpenShift to support the persistent storage for containers.

Red Hat OpenShift Container Storage can be implemented for the OpenShift platform in two ways – *converged mode* or standalone *independent mode*.

In the *converged mode*, it is integrated with the OpenShift nodes such that the storage services are running next to the other node services. In other words, storage is embedded with the same nodes that are running the application workload containers. The advantage with this architecture is that the persistent storage for containers comes from the same cluster storage where the containers are running. This is a hyperconverged (HCI) architecture for implementing OpenShift.

In the *independent* mode, it runs on its own standalone cluster and the OpenShift nodes access the storage via the persistent volume mapping. In this mode, storage is separate to the compute nodes. The storage can be scaled independently of the compute nodes by adding more servers later to the storage cluster.

For more information on Red Hat OpenShift Container Storage, see: access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#install-config-persistent-storage-persistent-storage-glusterfs

Figure 13 gives a high-level overview of storage clusters in an OpenShift implementation.

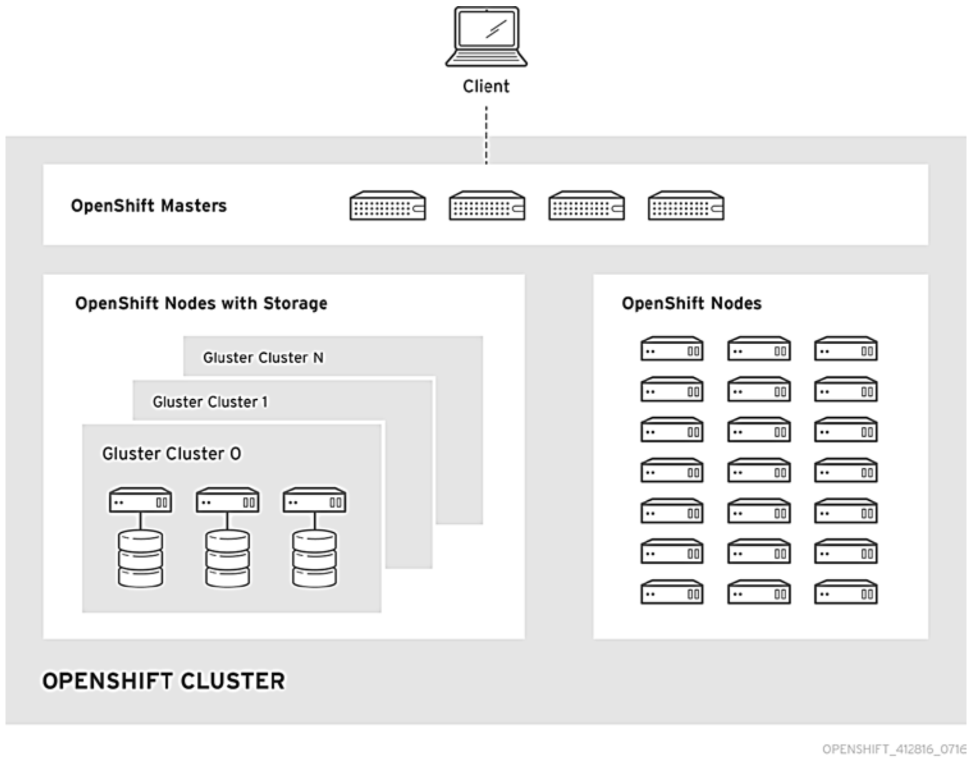


Figure 13. Gluster Storage for OpenShift

6.5 Networking

For OpenShift Container Platform deployment, 10Gbps networking is recommended as the choice for all cluster-wide communication for the core OpenShift services, virtual network implementation for container workloads, storage services access with Gluster FS, as well as all east-west traffic across the container workloads. In addition, the north-south traffic between the OpenShift environment and uplink into the customer (or campus) network can be implemented over the 10Gbps network.

There are three logical networks defined 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 14 shows the Red Hat OpenShift servers and the recommended network architecture.

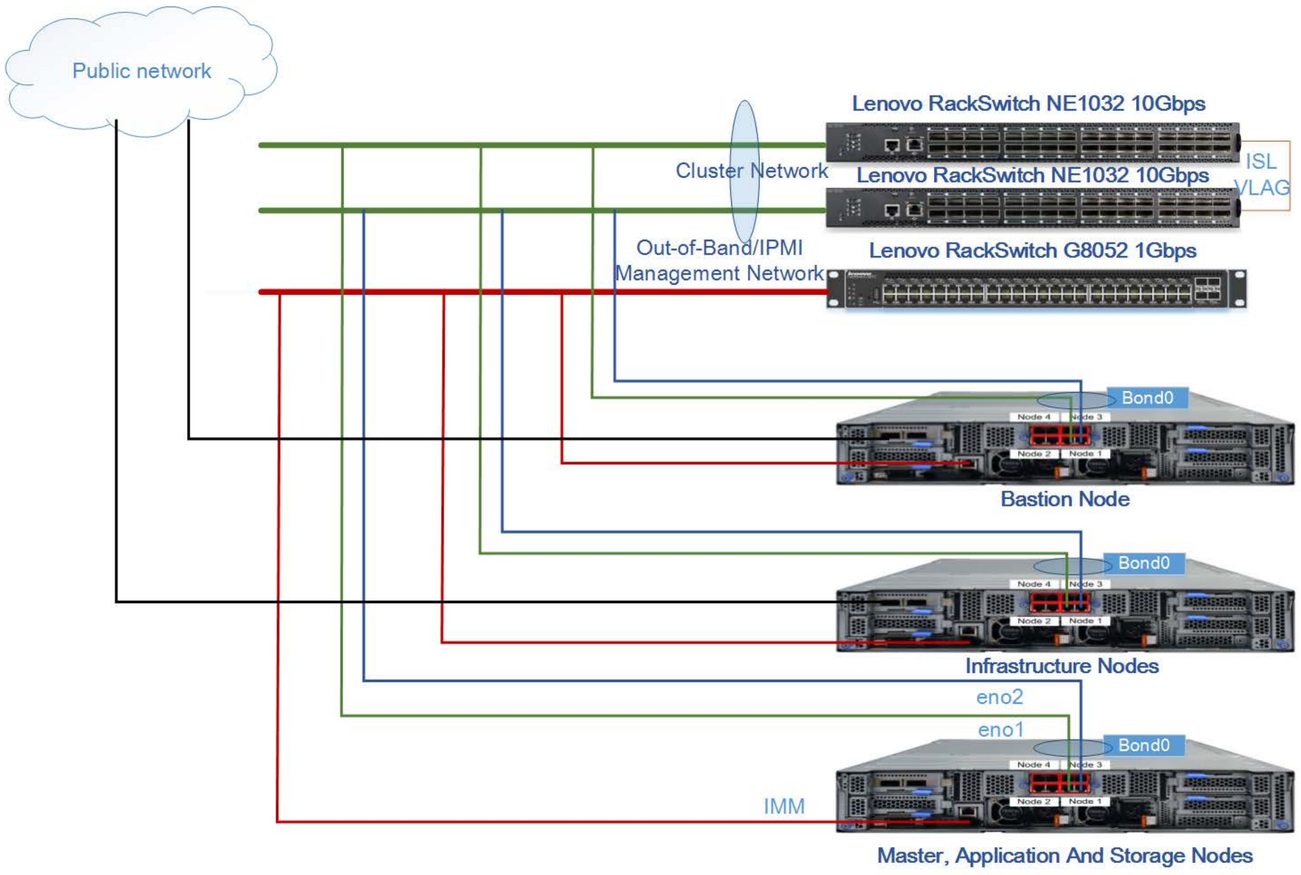

Public network
Lenovo RackSwitch NE1032 10Gbps
Cluster Network
Lenovo RackSwitch NE1032 10Gbps
ISL
VLAG
Out-of-Band/IPMI
Management Network
Lenovo RackSwitch G8052 1Gbps
Bond0
Bastion Node
Bond0
Infrastructure Nodes
eno2
eno1
Bond0
IMM
Master, Application And Storage Nodes

Figure 14. OpenShift Network Connectivity

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.

6.5.1 Hardware management network

For out-of-band management of the servers and initial cluster deployment over the network from the *bastion* node, use the 1Gbps management fabric via the Lenovo RackSwitch G8052. The Lenovo ThinkSystem rack servers have a dedicated 1GbE network port for the XCC interface. The XCC enables remote-management 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.

6.5.2 Network redundancy

The Lenovo OpenShift platform uses the 10 GbE network as the primary fabric for inter-node communication. Two Lenovo ThinkSystem NE1032 RackSwitch switches are used to provide redundant data layer communication and deliver maximum availability.

Figure 15 shows the redundant network architecture.

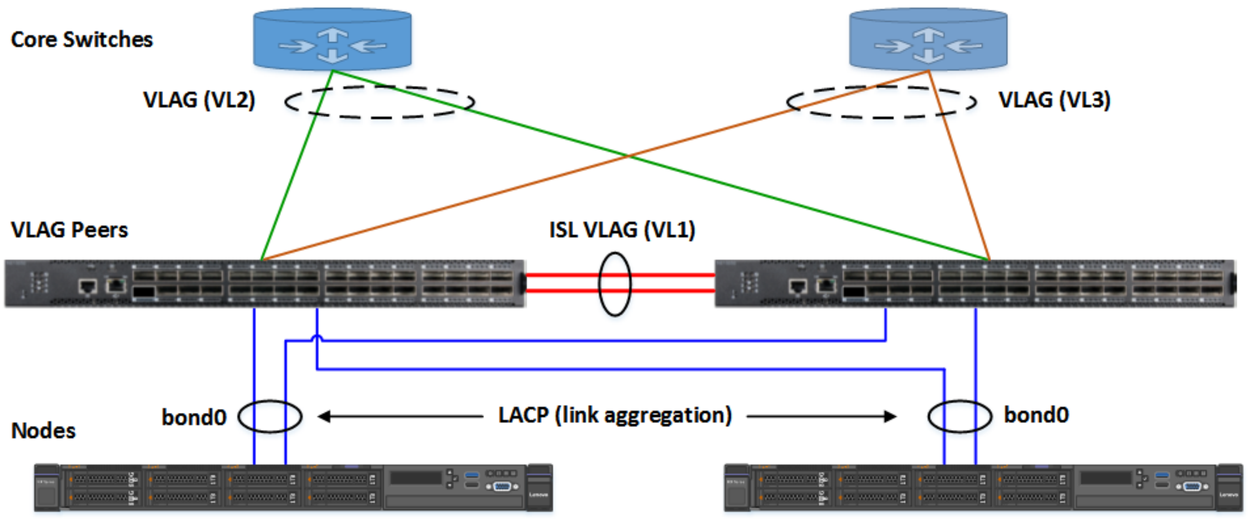


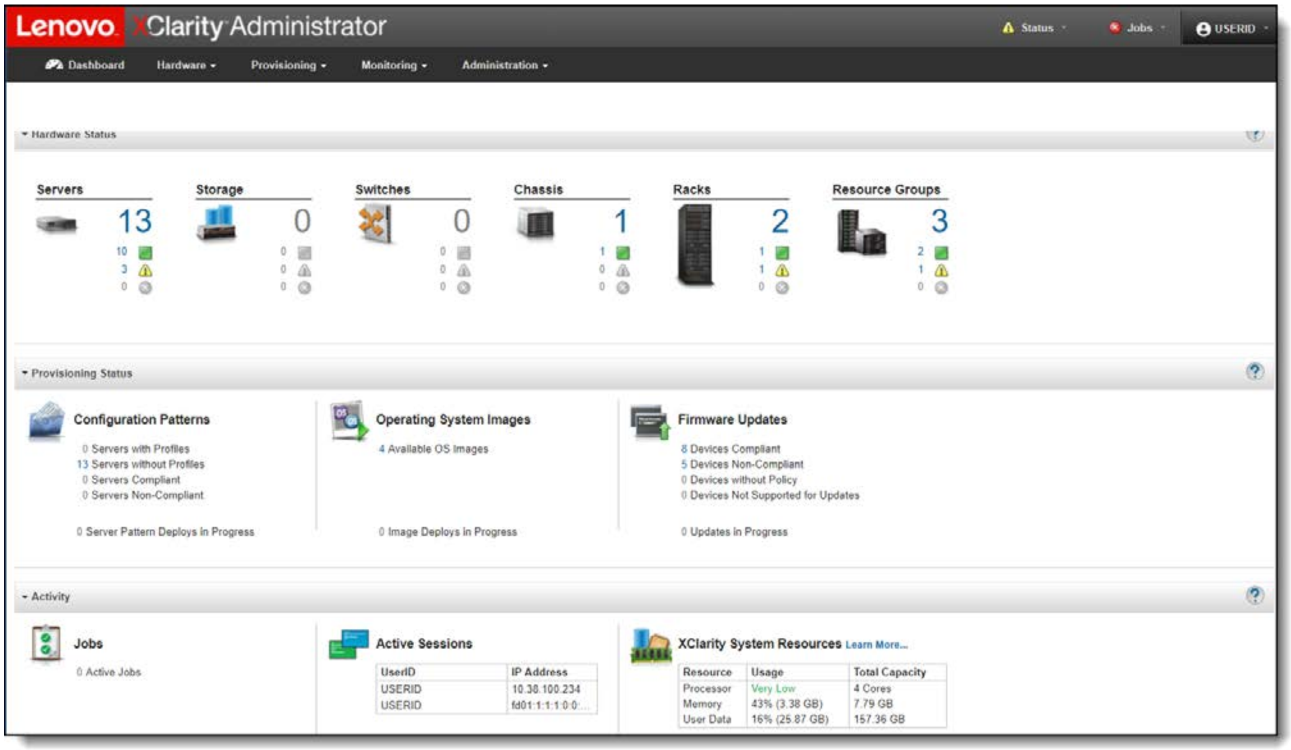
Figure 15. Redundant network architecture

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.

6.6 Systems management

In addition to in-band management via IPMI, the Lenovo XClarity Administrator software provides centralized resource management that reduces complexity, speeds up response, and enhances the availability of Lenovo® server systems and solutions.

The Lenovo XClarity Administrator provides agent-free hardware management for Lenovo's ThinkSystem® rack servers, System x® rack servers, and Flex System™ compute nodes and components, including the Chassis Management Module (CMM) and Flex System I/O modules. Figure 16 shows the Lenovo XClarity administrator interface, in which Flex System components and rack servers are managed and are seen on the dashboard. Lenovo XClarity Administrator is a virtual appliance that is quickly imported into a virtualized environment server configuration.



Servers	Storage	Switches	Chassis	Racks	Resource Groups
13	0	0	1	2	3
10 🟩	0	0	1 🟩	1 🟩	2 🟩
3 ⚠	0	0	0	1 ⚠	1 ⚠
0	0	0	0	0	0

Configuration Patterns
0 Servers with Profiles
13 Servers without Profiles
0 Servers Compliant
0 Servers Non-Compliant
0 Server Pattern Deploys in Progress

Operating System Images
4 Available OS Images
0 Image Deploys in Progress

Firmware Updates
8 Devices Compliant
5 Devices Non-Compliant
0 Devices without Policy
0 Devices Not Supported for Updates
0 Updates in Progress

Jobs
0 Active Jobs

Active Sessions

UserID	IP Address
USERID	10.38.100.234
USERID	fd01:1:1:1:0:0:...

XClarity System Resources

Resource	Usage	Total Capacity
Processor	Very Low	4 Cores
Memory	43% (3.38 GB)	7.79 GB
User Data	16% (25.87 GB)	157.36 GB

Figure 16. Lenovo XClarity Administrator Dashboard

For more information, see: [Lenovo XClarity Administrator Product Guide]

6.7 Deployment example

This deployment example was fully tested and verified by Lenovo. All of the deployment files are available from GitHub at the following location: github.com/lenovo/deployment-scripts-for-OpenShift-Platform.

The example uses 12 nodes as follows:

- 1 Bastion node
- 3 Master nodes
- 2 Infrastructure nodes
- 3 hyper-converged Application nodes with Red Hat OpenShift Container Storage to provide hyper-converged compute
- 3 storage-only Application nodes with Red Hat OpenShift Container Storage to provide an additional storage cluster (no compute)

This configuration represents a production grade OpenShift implementation that meets high-availability, redundancy, and scale requirements for enterprises. Additional Application nodes can be added to increase the available compute and storage capacity.

Table 3 provides a hardware configuration summary for this example deployment using the Lenovo ThinkSystem SD530 server the each node. The detailed server BOMs can be found in ”Appendix A: Lenovo bill of materials” on page 34.

Table 3. Node Hardware Configuration

OpenShift Node Role	**ThinkSystem SD530 server configuration**
Bastion Node Master Node Infrastructure Node	2x Intel Xeon Gold 6126 12C 125W 2.6GHz Processor 384GB memory (12x 32 GB) 2x ThinkSystem M.2 5100 480GB SATA 6Gbps Non-Hot-Swap SSD 1x ThinkSystem M.2 with Mirroring Enablement Kit 1x ThinkSystem Intel X710-DA2 PCIe 10Gb 2-Port SFP+ Ethernet Adapter
Application node (with Red Hat OpenShift Container Storage)	2x Intel Xeon Gold 6126 12C 125W 2.6GHz Processor 384GB memory (12x 32 GB) 2x ThinkSystem 2.5" Intel S4610 960GB Mainstream SATA 6Gb 4x ThinkSystem 2.5" 2TB 7.2K SATA 6Gb Hot Swap 512e HDD 1x ThinkSystem 430-8i SAS/SATA 12Gb Dense HBA 1x ThinkSystem Intel X710-DA2 PCIe 10Gb 2-Port SFP+ Ethernet Adapter

The configuration for the Bastion, Master and Infrastructures nodes is the same. This allows the role for a server to be easily changed. The configuration for all of the Application nodes is also the same, regardless of whether a particular server is used for hyperconverged compute or storage only.

Figure 17 shows the rack-level diagram of the hardware including the data and management switches.

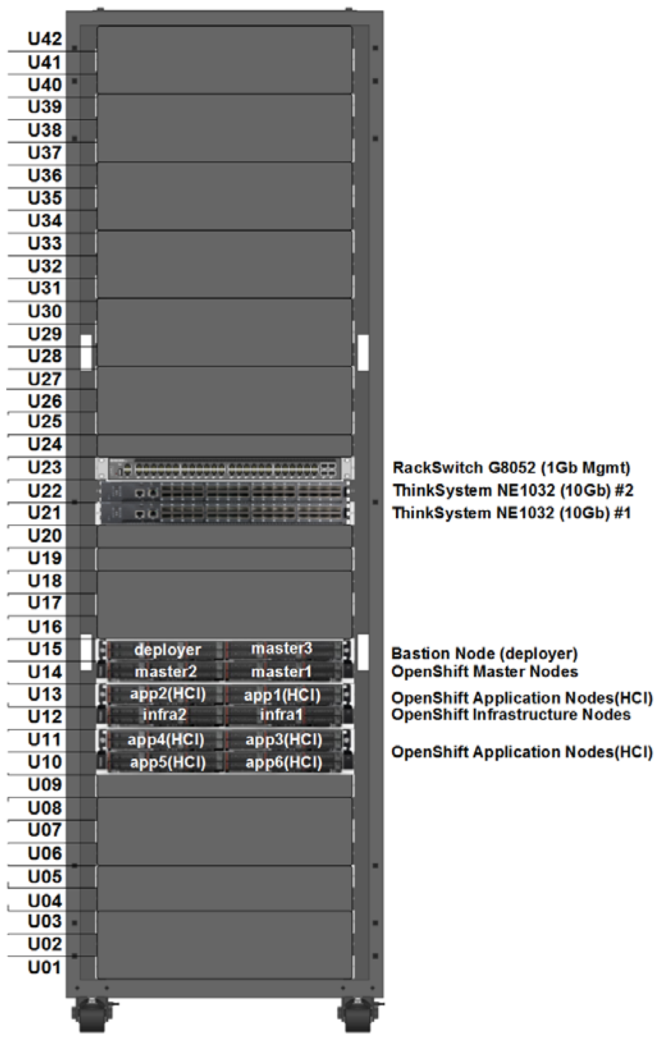


Figure 17. Rack level diagram of the OpenShift Container Platform implementation

6.7.1 Ansible inventory file

In order to install the OpenShift Container Platform cluster, an Ansible inventory file has to be created with the environment's description. The rest of this section explains the inventory file and provides the additional variables that are used for the automatic prerequisites and a Keepalived deployment.

The Ansible section below specifies the types of nodes that are used in an OpenShift Container Platform environment. Required groups are nodes, masters, and etcd. Optional groups are lb (for load balancing in multi-master clusters), local (which specifies the *Bastion* node), glusterfs (configure persistent volumes for application containers that require data persistence), glusterfs_registry (configure persistent volumes for docker-registry and application containers that require data persistence).

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

```
local
glusterfs
glusterfs_registry
```

The Ansible section below 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, many other authentication methods can be used such as LDAP, Keystone*, or GitHub* accounts. **`os_sdn_network_plugin_name`** specifies the SDN Open vSwitch plugin used in environment. For this example, **`redhat/openshift-ovs-multitenant`** provides isolation between OpenShift projects on the network level. The last three sections specify the storage backend type and its size for Docker Registry. In this solution, Docker Registry uses glusterfs server for Docker image storage.

```
[OSEv3:vars]
ansible_ssh_user=root
ansible_become=true
containerized=true
openshift_master_cluster_method=native
openshift_master_cluster_hostname=lb.ocp.example.local
openshift_master_cluster_public_hostname=openshift.ocp.example.com
openshift_master_default_subdomain=apps.ocp.example.com
openshift_master_cluster_ip=172.30.4.30
openshift_master_cluster_public_ip=10.240.202.188
openshift_master_portal_net=10.0.0.0/16
openshift_deployment_type=openshift-enterprise
openshift_release=v3.9
#openshift_image_tag=v3.9
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/users.htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$/1qMilNR$sURQoJkUyivYdW121JWK51'}

rhel_subscription_user=xxx
rhel_subscription_pass=xxx
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=200Gi
openshift_storage_glusterfs_registry_storageclass=true
```

The Ansible section below describes which servers act as OpenShift masters. For this example, 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.

```
[masters]
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
```

The Ansible section below describes which servers act as OpenShift nodes. For this example, 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. User application could be deployed on these servers when configuring parameter openshift_schedulable=true. In this RA, all node components are implemented as containers, which is determined by the **containerized=true** parameter.

```
[nodes]
master1.ocp.example.local openshift_schedulable=true containerized=true
openshift_public_ip=10.240.202.142 openshift_ip=172.30.4.9
openshift_hostname=master1.ocp.example.local ipmi=10.240.202.141 serial=J300A1KH
master2.ocp.example.local openshift_schedulable=true containerized=true
openshift_public_ip=10.240.202.144 openshift_ip=172.30.4.10
openshift_hostname=master2.ocp.example.local ipmi=10.240.202.143 serial=J300A1KP
master3.ocp.example.local openshift_schedulable=true containerized=true
openshift_public_ip=10.240.202.146 openshift_ip=172.30.4.11
openshift_hostname=master3.ocp.example.local ipmi=10.240.202.145 serial=J300A1KT
infra2.ocp.example.local openshift_node_labels="{'region': 'infra'}"
openshift_schedulable=true containerized=true openshift_public_ip=10.240.202.150
openshift_ip=172.30.4.8 openshift_hostname=infra2.ocp.example.local
ipmi=10.240.202.149 serial=J300A1KW
infra1.ocp.example.local openshift_node_labels="{'region': 'infra'}"
openshift_schedulable=true containerized=true openshift_public_ip=10.240.202.148
openshift_ip=172.30.4.7 openshift_hostname=infra1.ocp.example.local
ipmi=10.240.202.147 serial=J300A1KN
app1.ocp.example.local openshift_schedulable=true containerized=true
openshift_ip=172.30.4.2 openshift_hostname=app1.ocp.example.local
openshift_hostname_check=false ipmi=10.240.202.153 serial=J300A1KP
app0.ocp.example.local openshift_schedulable=true containerized=true
openshift_ip=172.30.4.1 openshift_hostname=app0.ocp.example.local  ipmi=10.240.202.151
serial=J300A1KM
```

```
app2.ocp.example.local openshift_schedulable=true containerized=true
openshift_ip=172.30.4.3 openshift_hostname=app2.ocp.example.local  ipmi=10.240.202.155
serial=J300A1KG
gluster3.ocp.example.local containerized=true openshift_schedulable=true
openshift_ip=172.30.4.6 openshift_hostname=gluster3.ocp.example.local
ipmi=10.240.202.173 serial=J300A1KV
gluster1.ocp.example.local containerized=true openshift_schedulable=true
openshift_ip=172.30.4.4 openshift_hostname=gluster1.ocp.example.local
ipmi=10.240.202.157 serial=J300A1KK
gluster2.ocp.example.local containerized=true openshift_schedulable=true
openshift_ip=172.30.4.5 openshift_hostname=gluster2.ocp.example.local
ipmi=10.240.202.159 serial=J300A1KR
```

The Ansible section below describes hosts that will run etcd instances. For this example, 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.

```
[etcd]
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
```

When **`openshift_master_cluster_method`** is set to **`native`**, then the Ansible section below specifies a host on which HAProxy load balancer will be installed and configured. For this example, 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.

```
[lb]
infra1.ocp.example.local openshift_hostname=infra1.ocp.example.local
openshift_ip=172.30.4.7 openshift_public_ip=10.240.202.148
infra2.ocp.example.local openshift_hostname=infra2.ocp.example.local
openshift_ip=172.30.4.8 openshift_public_ip=10.240.202.150
```

The Ansible section below describes which servers act as Red Hat OpenShift Container Storage nodes for storage backend coexisted with openshift compute components. This provides storage available for RedHat OpenShift users and can be used when storage for intense workloads is required.

```
[glusterfs]
app0.ocp.example.local glusterfs_ip=172.30.4.1 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf']"
app1.ocp.example.local glusterfs_ip=172.30.4.2 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf']"
```

```
app2.ocp.example.local glusterfs_ip=172.30.4.3 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf']"
```

The Ansible section below describes which servers act as another set of Red Hat OpenShift Container Storage nodes. For this example, three containerized Red Hat OpenShift Container Storage nodes are implemented on dedicated hosts. Additionally, the file specifies which of their IP addresses and disks act as GlusterFS volumes. This cluster is used as a default storage backend. It also provides a persistent volume for the OpenShift private Docker registry.

```
[glusterfs_registry]
gluster1.ocp.example.local glusterfs_ip=172.30.4.4 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf' ]"
gluster2.ocp.example.local glusterfs_ip=172.30.4.5 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf']"
gluster3.ocp.example.local glusterfs_ip=172.30.4.6 glusterfs_devices="[ '/dev/sdb',
'/dev/sdc', '/dev/sdd', '/dev/sde', '/dev/sdf']"
```

6.7.2 Software

For this example, the following software is needed:

- **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

In addition, the OpenShift Container Platform requires the following software packages:

- **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
- **GlusterFS** for storing application images and providing persistent volumes (PV) functionality

Table 4 lists the software versions used for this example deployment

Table 4. Software versions

Component	**Version**
Red Hat Enterprise Linux	7.4
OpenShift Container Platform	3.9
Docker	1.13.1
Ansible	2.6.0
rhel7/etcd	latest
openshift3/openvswitch	v3.9.40

Component	**Version**
openshift3/ose-haproxy-router	v3.9.40
openshift3/keepalived	1.0_ra
rhgs3/rhgs-gluster-block-prov-rhel7	latest

Each node is installed with RHEL 7.4 as the base operating system and the following resources:

- Minimum 40 GB hard disk space for the file system containing ***`/var/`***.
- Minimum 1 GB hard disk space for the file system containing ***`/usr/local/bin/`***.
- Minimum 1 GB hard disk space for the file system containing the system's temporary directory

6.7.3 Networking

The Lenovo ThinkSystem NE1032 RackSwitch with CNOS provides a simple, open, and programmable network infrastructure. In particular CNOS allows Ansible modules and this deployment example makes use of its management capabilities to implement automated network provisioning.

To obtain the submodule for the CNOS switch, execute the following git commands in the main directory of the Lenovo example repository:

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

The next step is to add the switch information to the Ansible inventory file as follows (use your own IP addresses).

```
[switches]
10.240.202.133 username=admin password=admin deviceType=NE1032
10.240.202.134 username=admin password=admin deviceType=NE1032
```

Then 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
```

Then go to the subdirectory containing the switch configuration component and run the playbook shown with the appropriate Python path:

```
$ cd src/cnos-configuration/roles/configure-networking
$ ansible-playbook configure-networking.yaml
```

After completing the playbook, the network interfaces on the two switches are configured correctly.

6.7.4 Automatic prerequisites installation

After the operating system has been installed and configured, the nodes need to 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 the remaining tasks can be executed automatically using the Ansible playbooks available at https://github.com/lenovo/deployment-scripts-for-OpenShift-Platform.

Based on information from the operating system deployment, prepare an Ansible inventory file and place it in the location /etc/ansible/hosts on the *Bastion* node. The hosts file for this deployment example is named hosts.example and is provided in the GitHub repository.

After that, clone the git repository onto the *Bastion* node:

```
$ git clone https://github.com/lenovo/deployment-scripts-for-OpenShift-Platform.git
$ cd deployment-scripts-for-OpenShift-Platform/src/prerequisites
```

In the inventory file, set up the following additional variables:

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

Finally start the Ansible playbook by entering the following command:

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

6.7.5 Automatic Keepalived deployment

The OpenShift Container Platform delivers two flavors of HAProxy load balancing software. The first flavor, which is 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 high availability (HA), maximum fault tolerance, and performance, this deployment example includes an additional package called Keepalived. It is open-source software distributed under the GPL license and is recognized by Red Hat as their recommended solution. Please see the following web site for more information: Red Hat Enterprise Linux documentation.

This example deployment uses both flavors of HAProxy instances, which are installed on both Infrastructure nodes. A single point of failure is eliminated is eliminated when used in conjunction with floating IP addresses provided by Keepalived,

Installation and configuration of this HA Solution can be performed manually or through a single command using an Ansible playbook. First the following variables must be defined in the Ansible inventory:

```
external_interface=enp6s0f0
external_netmask=255.255.250.128
external_gateway=10.240.202.129
external_vlan= 302
external_dns=114.114.114.114
internal_interface=bond0
internal_netmask=255.255.0.0
openshift_master_cluster_ip=172.30.4.30
openshift_master_cluster_public_ip=10.240.202.188
```

To deploy the Keepalived daemons using an Ansible playbook on Infrastructure nodes, enter following command inside the cloned GitHub repository:

```
$ su openshift
$ ansible-playbook \
    openshift-container-architecture/src/keepalived-multimaster/keepalived.yaml
```

6.7.6 OpenShift container platform installation

When the inventory file with the environment description is prepared and all prerequisites are configured, the OpenShift Container Platform install can be performed from the *Bastion* node.

For this deployment example, a containerized version of the OpenShift Container Platform is installed on servers. This installer image provides the same functionality as the RPM-based installer, but it runs in a containerized environment that provides all of its dependencies rather than being installed directly on the node. The only requirement to use it is the ability to run a container, and atomic packages are installed. The installer image can be used as a system container. System containers are stored and run outside of the traditional Docker service. This enables running the installer image from one of the target hosts without concern for the install restarting Docker on the host.

This install process is straightforward, and requires two steps:

First execute the prerequisites:

```
$ atomic install --system --set INVENTORY_FILE=/etc/ansible/hosts --storage=ostree
--set PLAYBOOK_FILE=/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
--set OPTS="-v" registry.access.redhat.com/openshift3/ose-ansible:v3.9
```

Second, deploy OpenShift platform：

```
$ atomic install –-system --storage=ostree  --set INVENTORY_FILE=/etc/ansible/hosts
--set PLAYBOOK_FILE=/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
--set OPTS="-v" registry.access.redhat.com/openshift3/ose-ansible:v3.9
```

After the installation process, the Ansible playbook should report no errors and the OpenShift Container Platform environment is set up. If needed, you can easily uninstall the environment with the following command:

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

Next user credentials should be created using the following commands:

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

Use the **htpasswd** command for each user account and propagate the users.htpasswd file to every OpenShift *master* node, into the ***/etc/origin/master/*** directory.

Lastly restart the API services on each *master* node using the command:

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

6.7.7 Deployment validation

The deployment should be validated before it is used.

First, log on to one of the OpenShift *master* nodes and check that all nodes are connected to the cluster using the commands:

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

Here is some example output from the command:

```
NAME                        STATUS  ROLES      AGE    VERSION
app0.ocp.example.local      Ready   compute    26d    v1.9.1+a0ce1bc657
app1.ocp.example.local      Ready   compute    26d    v1.9.1+a0ce1bc657
app2.ocp.example.local      Ready   compute    26d    v1.9.1+a0ce1bc657
gluster1.ocp.example.local Ready   compute    26d    v1.9.1+a0ce1bc657
gluster2.ocp.example.local Ready   compute    26d    v1.9.1+a0ce1bc657
gluster3.ocp.example.local Ready   compute    26d    v1.9.1+a0ce1bc657
infra1.ocp.example.local   Ready   <none>     26d    v1.9.1+a0ce1bc657
infra2.ocp.example.local   Ready   <none>     26d    v1.9.1+a0ce1bc657
master1.ocp.example.local  Ready   master     26d    v1.9.1+a0ce1bc657
master2.ocp.example.local  Ready   master     26d    v1.9.1+a0ce1bc657
master3.ocp.example.local  Ready   master     26d    v1.9.1+a0ce1bc657
```

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 using the following command to verify the etcd state:

```
$ 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
```

Here is some example output from the command:

```
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
```

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.

To further inspect and verify all the components of the OpenShift Container Platform cluster, use the command:

```
$ oc get all
```

All the listed items should have a **Running** status.

At a final verification step, log on to the OpenShift Container Platform web console using the following URL address: https://openshift.ocp.example.com:8443 and display the OpenShift container catalog. Figure 18 shows an example.

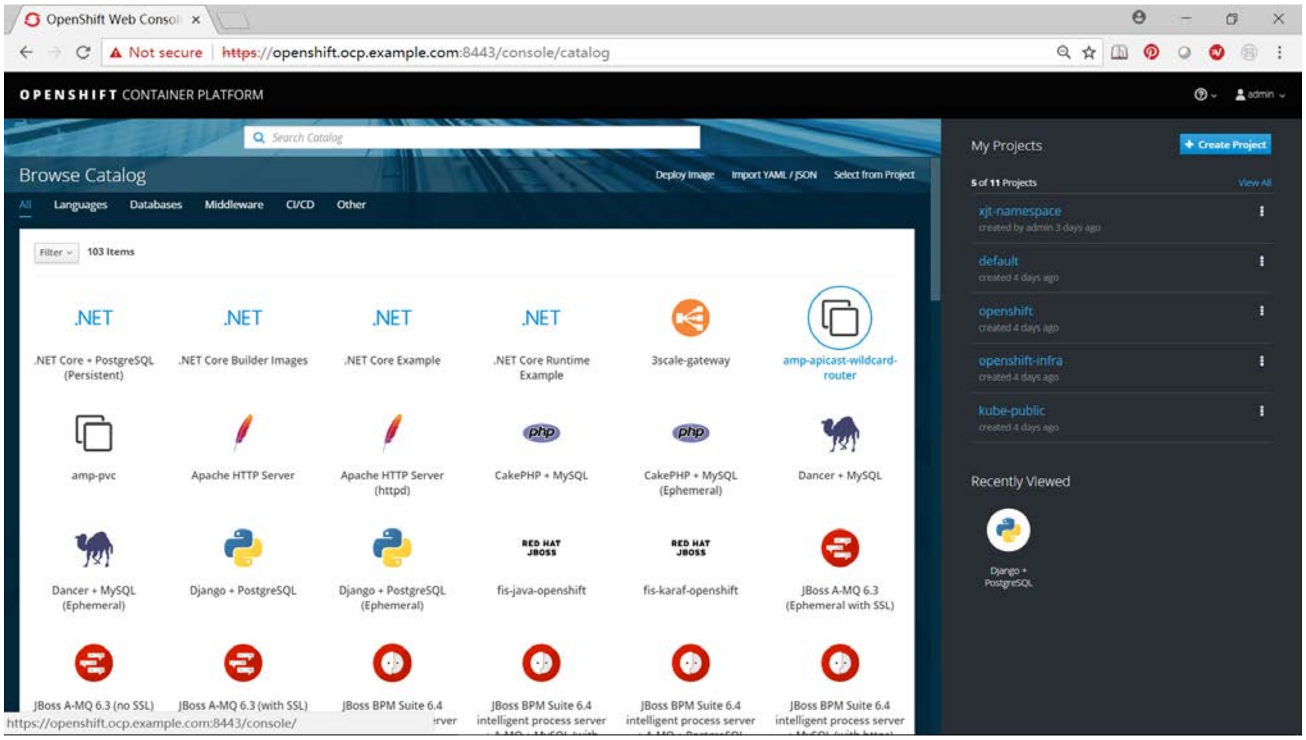


Figure 18. Example Catalog for OpenShift Container Platform

Figure 19 shows other OpenShift application components.

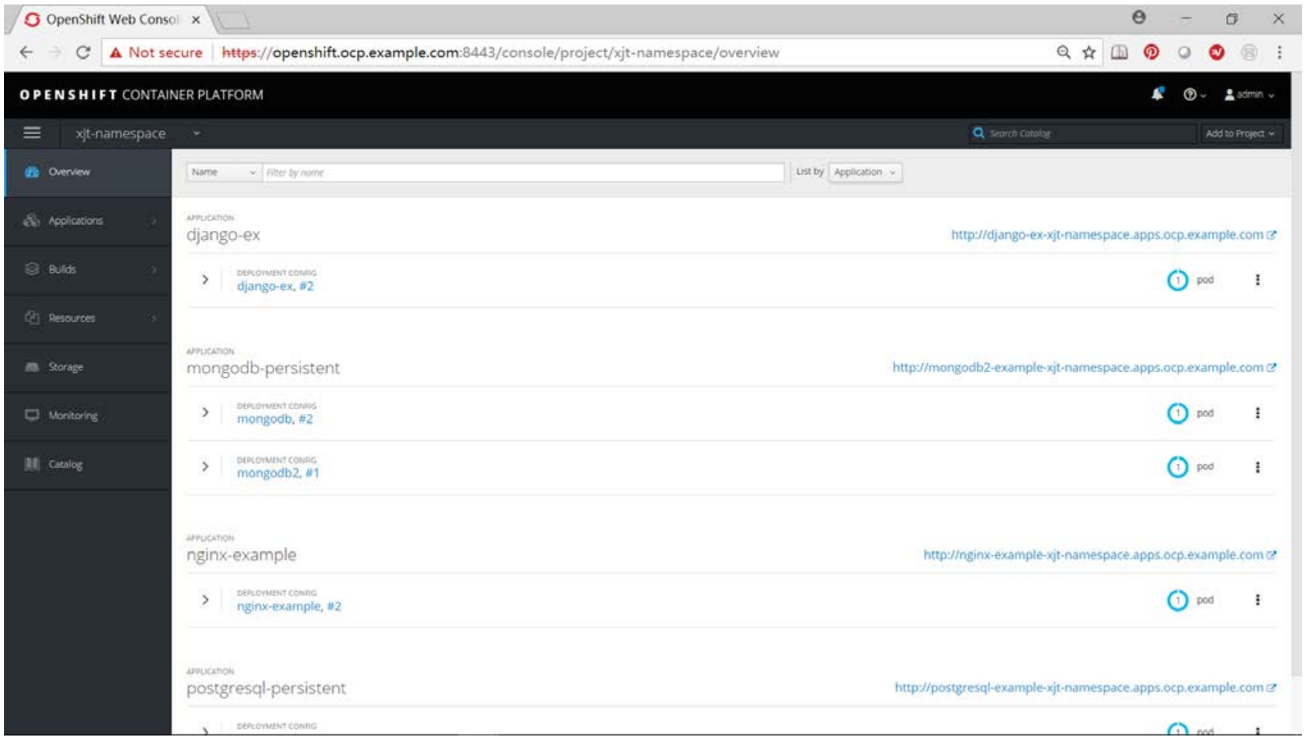


Figure 19. OpenShift Application Components

7 Appendix A: Lenovo bill of materials

This appendix contains the bill of materials (BOMs) for different configurations of hardware for OpenShift deployments. There are sections for servers and networking.

7.1 Server BOM

For OpenShift, servers with same configurations are deployed as different nodes (Bastion, Master, Infrastructure, App, and Storage).

Lenovo ThinkSystem SD530 (Bastion Node, Master Node, Infrastructure Node)

Code	Description	Quantity
7X21CTO1WW	Node : ThinkSystem SD530 - 3yr Warranty	1
AUXN	ThinkSystem SD530 Computing Node	1
AWEX	Intel Xeon Gold 6126 12C 125W 2.6GHz Processor	2
AUND	ThinkSystem 32GB TruDDR4 2666 MHz (2Rx4 1.2V) RDIMM	12
AUYG	ThinkSystem SD530 3x2 SAS/SATA BP	1
AUMV	ThinkSystem M.2 with Mirroring Enablement Kit	1
B11V	ThinkSystem M.2 5100 480GB SATA 6Gbps Non-Hot Swap SSD	2
AUKX	ThinkSystem Intel X710-DA2 PCIe 10Gb 2-Port SFP+ Ethernet Adapter	2
A51P	2m Passive DAC SFP+ Cable	1
B0MJ	Feature Enable TPM 1.2	1
AVUT	ThinkSystem XClarity Controller Standard to Advanced Upgrade	1
B0ML	Feature Enable TPM on MB	1

Lenovo ThinkSystem SD530 (Application node with storage)

Code	Description	Quantity
7X21CTO1WW	Node : ThinkSystem SD530 - 3yr Warranty	1
AUXN	ThinkSystem SD530 Computing Node	1
AWEX	Intel Xeon Gold 6126 12C 125W 2.6GHz Processor	2
AUND	ThinkSystem 32GB TruDDR4 2666 MHz (2Rx4 1.2V) RDIMM	12
AUYG	ThinkSystem SD530 3x2 SAS/SATA BP	1
5977	Select Storage devices - no configured RAID required	1
B0SS	ThinkSystem 430-8i SAS/SATA 12Gb Dense HBA	1
B49N	ThinkSystem 2.5" Intel S4610 960GB Mainstream SATA 6Gb HS SSD	2
AUUJ	ThinkSystem 2.5" 2TB 7.2K SATA 6Gb Hot Swap 512e HDD	4
AUKX	ThinkSystem Intel X710-DA2 PCIe 10Gb 2-Port SFP+ Ethernet Adapter	2
A51P	2m Passive DAC SFP+ Cable	1
B0MJ	Feature Enable TPM 1.2	1
AVUT	ThinkSystem XClarity Controller Standard to Advanced Upgrade	1
B0ML	Feature Enable TPM on MB	1

ThinkSystem D2 Enclosure

Code	Description	Quantity
7X20CTO1WW	Chassis : ThinkSystem D2 Enclosure -3yr Warranty	1
AUXM	ThinkSystem D2 Enclosure	1
AUY8	ThinkSystem D2 4-slot x16 Shuttle	1
AUY9	ThinkSystem D2 10Gb 8 port EIOM SFP+	1
AUZ2	ThinkSystem D2 2000W Platinum PSU	2
6311	2.8m, 10A/100-250V, C13 to C14 Jumper Cord	2
AUYC	ThinkSystem D2 Slide Rail	1
A51P	2m Passive DAC SFP+ Cable	8
AVFZ	1.5m Green Cat6 Cable	1
AVR1	ThinkSystem Single Ethernet Port SMM	1
AVT5	ThinkSystem D2 Chassis Agency Label	1
AVD0	System Document	1
A2N6	Chassis Not Integrated With Planar	1
AUXQ	ThinkSystem D2 NODE Dummy Filler	3
AUYT	ThinkSystem D2 Chassis Package	1

Lenovo ThinkSystem SR630 (Bastion Node, Master Node, Infrastructure Node)

Code	Description	Quantity
7X02CTO1WW	Server : ThinkSystem SR630 - 3yr Warranty	1
AUW0	ThinkSystem SR630 2.5" Chassis with 8 Bays	1
AWEL	Intel Xeon Gold 6126 12C 125W 2.6GHz Processor	2
AUND	ThinkSystem 32GB TruDDR4 2666 MHz (2Rx4 1.2V) RDIMM	12
AUWB	ThinkSystem SR530/SR630/SR570 2.5" SATA/SAS 8-Bay Backplane	1
AUMV	ThinkSystem M.2 with Mirroring Enablement Kit	1
B11V	ThinkSystem M.2 5100 480GB SATA 6Gbps Non-Hot Swap SSD	2
AUKK	ThinkSystem 10Gb 4 Port SFP+ LOM	1
A51P	2m Passive DAC SFP+ Cable	3
AVWA	ThinkSystem 750W (230/115V) Platinum Hot-Swap Power Supply	2
6400	2.8m, 13A/100-250V, C13 to C14 Jumper Cord	2
B0MJ	Feature Enable TPM 1.2	1
B0ML	Feature Enable TPM on MB	1
AUWG	Lenovo ThinkSystem 1U VGA Filler	1
AUW3	Lenovo ThinkSystem Mainstream MB - 1U	1
AUW7	ThinkSystem SR630 4056 Fan Module	2
AULP	ThinkSystem 1U CPU Heatsink	2

Lenovo ThinkSystem SR630 (Application node with storage)

Code	Description	Quantity
7X02CTO1WW	Server : ThinkSystem SR630 - 3yr Warranty	1
AUW0	ThinkSystem SR630 2.5" Chassis with 8 Bays	1
AWEL	Intel Xeon Gold 6126 12C 125W 2.6GHz Processor	2
AUND	ThinkSystem 32GB TruDDR4 2666 MHz (2Rx4 1.2V) RDIMM	12
AUWB	ThinkSystem SR530/SR630/SR570 2.5" SATA/SAS 8-Bay Backplane	1
5977	Select Storage devices - no configured RAID required	1
AUNL	ThinkSystem 430-8i SAS/SATA 12Gb HBA	1
B49N	ThinkSystem 2.5" Intel S4610 960GB Mainstream SATA 6Gb HS SSD	2
AUUJ	ThinkSystem 2.5" 2TB 7.2K SATA 6Gb Hot Swap 512e HDD	4
AUKK	ThinkSystem 10Gb 4 Port SFP+ LOM	1
A51P	2m Passive DAC SFP+ Cable	3
AVWA	ThinkSystem 750W (230/115V) Platinum Hot-Swap Power Supply	2
6400	2.8m, 13A/100-250V, C13 to C14 Jumper Cord	2
B0MJ	Feature Enable TPM 1.2	1
B0ML	Feature Enable TPM on MB	1
AUWG	Lenovo ThinkSystem 1U VGA Filler	1
AUW3	Lenovo ThinkSystem Mainstream MB - 1U	1
AUW7	ThinkSystem SR630 4056 Fan Module	2
AULP	ThinkSystem 1U CPU Heatsink	2

7.2 Networking BOM

Lenovo ThinkSystem NE1032 Switch

Code	Description	Quantity
7159HD1	Switch : Lenovo ThinkSystem NE1032 RackSwitch (Rear to Front)	1
AU3A	Lenovo ThinkSystem NE1032 RackSwitch (Rear to Front)	1
A1PH	1m Passive DAC SFP+ Cable	1
6204	2.8m, 10A/100-250V, C13 to IEC 320-C20 Rack Power Cable	2

Lenovo RackSwitch G8052

Code	Description	Quantity
7159G52	Lenovo System Networking RackSwitch G8052 (Rear to Front)	1
6201	1.5m, 10A/100-250V, C13 to IEC 320-C14 Rack Power Cable	2
3802	1.5m Blue Cat5e Cable	3
A3KP	Lenovo System Networking Adjustable 19" 4 Post Rail Kit	1

Resources

- OpenShift Architecture Overview

docs.openshift.com/container-platform/3.9/architecture/

- Architecture of the Red Hat OpenShift Container Platform

access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/architecture/architecture-index#arch-index-what-is-the-architecture

- OpenShift Container Platform

redhat.com/en/technologies/cloud-computing/openshift

- Kubernetes

kubernetes.io/ and kubernetes.io/docs/tutorials/kubernetes-basics/

- Docker

docker.com/

- OpenShift containerized installation

docs.openshift.com/container-platform/3.9/install_config/install/advanced_install.html

- Red Hat OpenShift Container Storage

access.redhat.com/products/red-hat-openshift-container-storage

Trademarks and special notices