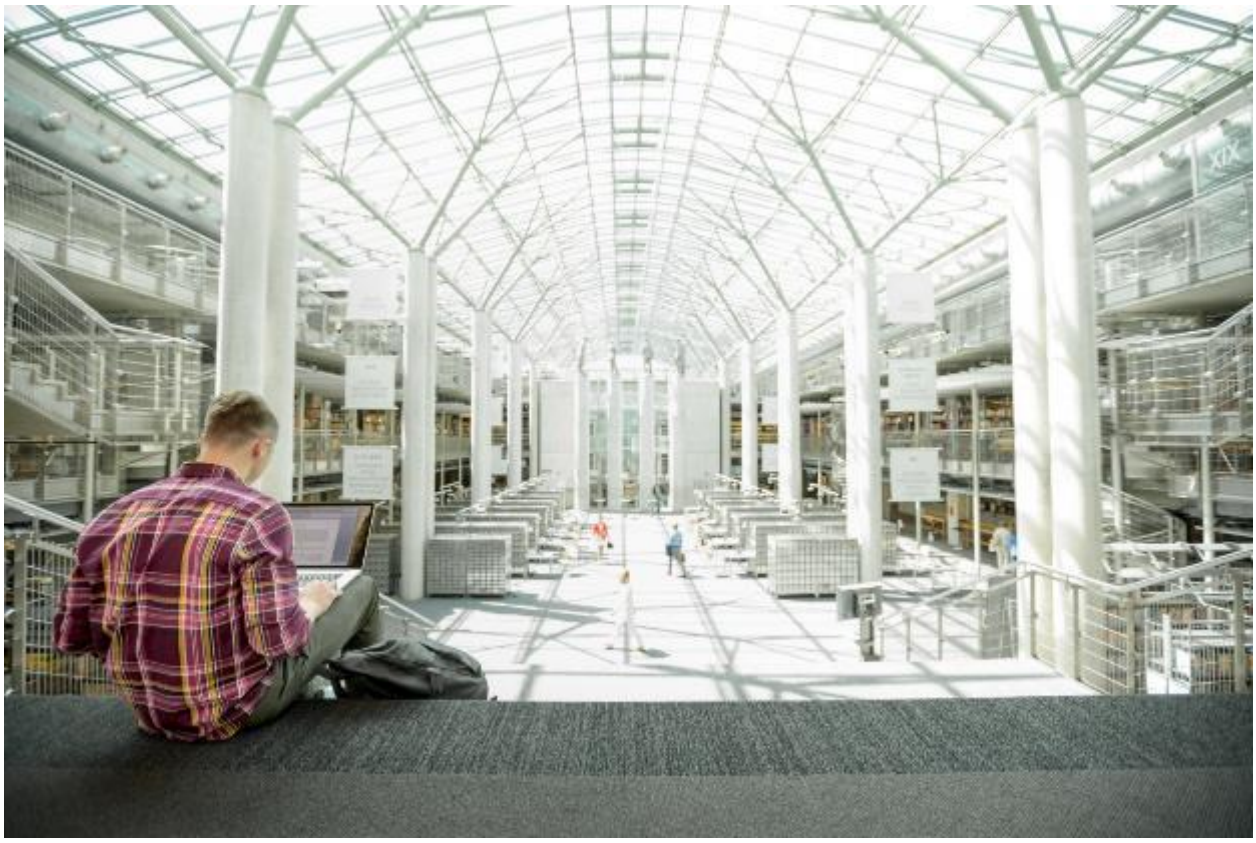


Cisco UCS Infrastructure with Red Hat OpenShift Container Platform on VMware vSphere

Design and Deployment Guide for Cisco UCS Infrastructure for Red Hat OpenShift Container Platform 3.9 and VMware vSphere 6.7 on Cisco UCS Manager 3.2, Cisco UCS M5 B-Series, and Cisco UCS M5 C-Series Servers

Last Updated: December 13, 2018

CISCO VALIDATED DESIGN

About the Cisco Validated Design Program

The Cisco Validated Design (CVD) program consists of systems and solutions designed, tested, and documented to facilitate faster, more reliable, and more predictable customer deployments. For more information, see:

http://www.cisco.com/go/designzone.

Table of Contents

Executive Summary

Cisco Validated Designs are the foundation of systems design and the centerpiece of facilitating complex customer deployments. The validated designs incorporate products and technologies into a broad portfolio of Enterprise, Service Provider, and Commercial systems that are designed, tested, and fully documented to help ensure faster, reliable, consistent, and more predictable customer deployments.

Cisco's converge infrastructures integrate systems that can make IT operations more cost-effective, efficient, and agile by integrating systems with Cisco Unified Computing System. These validated converged architectures accelerate customer's success by reducing risk with strategic guidance and expert advice.

Cisco leads in integrated systems and offers diversified portfolio of converged infrastructure of solutions. They integrate wide range of technologies and products into cohesive validated and supported solutions to address business needs of Cisco's customers. Furthermore, these converged infrastructures have been previously designed, tested, and validated with VMware vSphere environment.

The recommended solution architecture offers Red Hat OpenShift platform 3.9 on VMware vSphere 6.7 based on Cisco validated converged infrastructure. To learn more about datacenter virtualization on Cisco's validated converged infrastructure, see: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/converged-infrastructure/index.html.

Solution Overview

Introduction

Deployment-centric application platform and DevOps initiatives are driving benefits for organizations in their digital transformation journey. Though still early in maturity, Docker format container packaging and Kubernetes container orchestration are emerging to cater to the rapid digital transformation. Docker format container packaging and Kubernetes container orchestration are emerging as industry standards for state-of-the-art PaaS solutions.

Containers have brought a lot of excitement and value to IT by establishing predictability between building and running applications. Developers can trust and know that their application will perform the same when it's run on a production environment as it did when it was built, while operations and admins have the tools to operate and maintain applications seamlessly.

Red Hat® OpenShift® Container Platform provides a set of container-based open source tools enabling digital transformation, which accelerates application development while making optimal use of infrastructure. Professional developers utilize fine-grained control of all aspects of the application stack, with application configurations enabling rapid response to unforeseen events. Availability of highly secure operating systems assists in standing up an environment capable of withstanding continuously changing security threats, helping deployment with highly secure applications.

Red Hat OpenShift Container Platform helps organizations use the cloud delivery model and simplify continuous delivery of applications and services on Red Hat OpenShift Container Platform, the cloud-native way. Built on proven open source technologies, Red Hat OpenShift Container Platform also provides development teams multiple modernization options to enable a smooth transition to microservices architecture and the cloud for existing traditional applications.

Cisco Unified Computing System™ (Cisco UCS®) servers adapt to meet rapidly changing business needs, including just-in-time deployment of new computing resources to meet requirements and improve business outcomes. With Cisco UCS, you can tune your environment to support the unique needs of each application while powering all your server workloads on a centrally managed, highly scalable system. Cisco UCS brings the flexibility of non-virtualized and virtualized systems in a way that no other server architecture can, lowering costs and improving your return on investment (ROI).

Cisco UCS M5 servers built on Intel's powerful Intel® Xeon Scalable processors are unified yet modular, scalable, high-performing, built on infrastructure-as-code for powerful integrations and continuous delivery of distributed applications.

Cisco, Intel and Red Hat have joined hands to develop a best-in-class solution for delivering PaaS solution to the enterprise with ease. And also, to provide the ability to develop, deploy, and manage containers in an on-premises, Private/ Public cloud environments by bringing automation to the table with a robust platform such as Red Hat OpenShift Container Platform.

Solution Benefits

Some of the key benefits of this solution include:

- Red Hat OpenShift
  - Strong, role-based access controls, with integrations to enterprise authentication systems.
  - Powerful, web-scale container orchestration and management with Kubernetes.

  - Integrated Red Hat Enterprise Linux® Atomic Host, optimized for running containers at scale with Security-Enhanced Linux (SELinux) enabled for strong isolation.
  - Integration with public and private registries.
  - Integrated CI/CD tools for secure DevOps practices.
  - A new model for container networking.
  - Modernize application architectures toward microservices.
  - Adopt a consistent application platform for hybrid cloud deployments.
  - Support for remote storage volumes.
  - Persistent storage for stateful cloud-native containerized applications.
- Cisco UCS
  - Reduced datacenter complexities through Cisco UCS infrastructure with a single management control plane for hardware lifecycle management.
  - Easy to deploy and scale the solution.
  - Superior scalability and high-availability.
  - Compute form factor agnostic.
  - Better response with optimal ROI.
  - Optimized hardware footprint for production and dev/test deployments.

Audience

The audience for this document includes, but is not limited to, sales engineers, field consultants, professional services, IT managers, partner engineers, IT architects, and customers who want to take advantage of an infrastructure that is built to deliver IT efficiency and enable IT innovation. The reader of this document is expected to have the necessary training and background to install and configure Red Hat Enterprise Linux, Cisco Unified Computing System, and Cisco Nexus Switches, Enterprise storage sub-systems, VMware vSphere. Furthermore, knowledge of container platform preferably Red Hat OpenShift Container Platform is required. External references are provided where applicable and familiarity with these documents is highly recommended.

Purpose of this Document

This document highlights the benefits of using Cisco UCS M5 servers for Red Hat OpenShift Container Platform 3.9 on VMware vSphere 6.7 to efficiently deploy, scale, and manage a production-ready application container environment for enterprise customers. This document focuses design choices and best practices of deploying Red Hat OpenShift container platform on converged infrastructure comprised of Cisco UCS, Nexus, and VMware.

What's New in this Release?

In this solution Red Hat OpenShift Platform 3.9 is validated on Cisco UCS with VMware vSphere. Red Hat OpenShift Container platform nodes such as master, infrastructure, and application nodes are running in VMware vSphere virtualized environment while leveraging VMware HA cluster. Furthermore, GlusterFS is running on bare-metal environment on Cisco UCS C240 M5 which can also be utilized for provisioning containers on bare-metal.

Technology Overview

This section provides a brief introduction of the various hardware/ software components used in this solution.

Cisco Unified Computing System

The Cisco Unified Computing System is a next-generation solution for blade and rack server computing. The system integrates a low-latency; lossless 10 Gigabit Ethernet unified network fabric with enterprise-class, x86-architecture servers. The system is an integrated, scalable, multi-chassis platform in which all resources participate in a unified management domain. The Cisco Unified Computing System accelerates the delivery of new services simply, reliably, and securely through end-to-end provisioning and migration support for both virtualized and non-virtualized systems. Cisco Unified Computing System provides:

- Comprehensive Management
- Radical Simplification
- High Performance

The Cisco Unified Computing System consists of the following components:

- Compute–The system is based on an entirely new class of computing system that incorporates rack mount and blade servers based on Intel® Xeon® scalable processors product family.
- Network–The system is integrated onto a low-latency, lossless, 40-Gbps unified network fabric. This network foundation consolidates Local Area Networks (LAN's), Storage Area Networks (SANs), and high-performance computing networks which are separate networks today. The unified fabric lowers costs by reducing the number of network adapters, switches, and cables, and by decreasing the power and cooling requirements.
- Virtualization–The system unleashes the full potential of virtualization by enhancing the scalability, performance, and operational control of virtual environments. Cisco security, policy enforcement, and diagnostic features are now extended into virtualized environments to better support changing business and IT requirements.
- Storage access–The system provides consolidated access to both SAN storage and Network Attached Storage (NAS) over the unified fabric. It is also an ideal system for Software Defined Storage (SDS). Combining the benefits of single framework to manage both the compute and Storage servers in a single pane, Quality of Service (QOS) can be implemented if needed to inject IO throttling in the system. In addition, the server administrators can pre-assign storage-access policies to storage resources, for simplified storage connectivity and management leading to increased productivity. In addition to external storage, both rack and blade servers have internal storage which can be accessed through built-in hardware RAID controllers. With storage profile and disk configuration policy configured in Cisco UCS Manager, storage needs for the host OS and application data gets fulfilled by user defined RAID groups for high availability and better performance.
- Management–the system uniquely integrates all system components to enable the entire solution to be managed as a single entity by the Cisco UCS Manager. The Cisco UCS Manager has an intuitive graphical user interface (GUI), a command-line interface (CLI), and a powerful scripting library module for Microsoft PowerShell built on a robust application programming interface (API) to manage all system configuration and operations.

Cisco Unified Computing System (Cisco UCS) fuses access layer networking and servers. This high-performance, next-generation server system provides a data center with a high degree of workload agility and scalability.

Cisco UCS Manager

Cisco Unified Computing System Manager (Cisco UCS Manager) provides unified, embedded management for all software and hardware components in Cisco UCS. Using Single Connect technology, it manages, controls, and administers multiple chassis for thousands of virtual machines. Administrators use the software to manage the entire Cisco Unified Computing System as a single logical entity through an intuitive GUI, a command-line interface (CLI), or an XML API. The Cisco UCS Manager resides on a pair of Cisco UCS 6300 Series Fabric Interconnects using a clustered, active-standby configuration for high-availability.

Cisco UCS Manager offers unified embedded management interface that integrates server, network, and storage. Cisco UCS Manager performs auto-discovery to detect inventory, manage, and provision system components that are added or changed. It offers comprehensive set of XML API for third part integration, exposes 9000 points of integration and facilitates custom development for automation, orchestration, and to achieve new levels of system visibility and control.

Service profiles benefit both virtualized and non-virtualized environments and increase the mobility of non-virtualized servers, such as when moving workloads from server to server or taking a server offline for service or upgrade. Profiles can also be used in conjunction with virtualization clusters to bring new resources online easily, complementing existing virtual machine mobility.

For more information about Cisco UCS Manager Information,
see: http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-manager/index.html.

Cisco UCS Fabric Interconnects

The Fabric interconnects provide a single point for connectivity and management for the entire system. Typically deployed as an active-active pair, the system's fabric interconnects integrate all components into a single, highly-available management domain controlled by Cisco UCS Manager. The fabric interconnects manage all I/O efficiently and securely at a single point, resulting in deterministic I/O latency regardless of a server or virtual machine's topological location in the system.

Cisco UCS 6300 Series Fabric Interconnects support the bandwidth up to 2.43-Tbps unified fabric with low-latency, lossless, cut-through switching that supports IP, storage, and management traffic using a single set of cables. The fabric interconnects feature virtual interfaces that terminate both physical and virtual connections equivalently, establishing a virtualization-aware environment in which blade, rack servers, and virtual machines are interconnected using the same mechanisms. The Cisco UCS 6332-16UP is a 1-RU Fabric Interconnect that features up to 40 universal ports that can support 24 40-Gigabit Ethernet, Fiber Channel over Ethernet, or native Fiber Channel connectivity. In addition to this it supports up to 16 1- and 10-Gbps FCoE or 4-, 8- and 16-Gbps Fibre Channel unified ports.

Figure 1 Cisco UCS Fabric Interconnect 6332-16UP

For more information, see: https://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-6332-16up-fabric-interconnect/index.html.

Cisco UCS 5108 Blade Server Chassis

The Cisco UCS 5100 Series Blade Server Chassis is a crucial building block of the Cisco Unified Computing System, delivering a scalable and flexible blade server chassis. The Cisco UCS 5108 Blade Server Chassis is six rack units (6RU) high and can mount in an industry-standard 19-inch rack. A single chassis can house up to eight half-width Cisco UCS B-Series Blade Servers and can accommodate both half-width and full-width blade form factors. Four single-phase, hot-swappable power supplies are accessible from the front of the chassis. These power supplies are 92 percent efficient and can be configured to support non-redundant, N+ 1 redundant and grid-redundant configurations. The rear of the chassis contains eight hot-swappable fans, four power connectors (one per power supply), and two I/O bays for Cisco UCS 2304 Fabric Extenders. A passive mid-plane provides multiple 40 Gigabit Ethernet connections between blade serves and fabric interconnects. The Cisco UCS 2304 Fabric Extender has four 40 Gigabit Ethernet, FCoE-capable, Quad Small Form-Factor Pluggable (QSFP+) ports that connect the blade chassis to the fabric interconnect. Each Cisco UCS 2304 can provide one 40 Gigabit Ethernet ports connected through the midplane to each half-width slot in the chassis, giving it a total eight 40G interfaces to the compute. Typically configured in pairs for redundancy, two fabric extenders provide up to 320 Gbps of I/O to the chassis.

For more information, see: http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-5100-series-blade-server-chassis/index.html.

Cisco UCS B200 M5 Blade Server

The Cisco UCS B200 M5 Blade Server has the following:

- Up to two Intel Xeon Scalable CPUs with up to 28 cores per CPU
- 24 DIMM slots for industry-standard DDR4 memory at speeds up to 2666 MHz, with up to 3 TB of total memory when using 128-GB DIMMs
- Modular LAN On Motherboard (mLOM) card with Cisco UCS Virtual Interface Card (VIC) 1340, a 2-port, 40 Gigabit Ethernet, Fibre Channel over Ethernet (FCoE)–capable mLOM mezzanine adapter
- Optional rear mezzanine VIC with two 40-Gbps unified I/O ports or two sets of 4 x 10-Gbps unified I/O ports, delivering 80 Gbps to the server; adapts to either 10- or 40-Gbps fabric connections
- Two optional, hot-pluggable, Hard-Disk Drives (HDDs), Solid-State Disks (SSDs), or NVMe 2.5-inch drives with a choice of enterprise-class RAID or pass-through controllers

Figure 2 Cisco UCS B200 M5 Blade Server

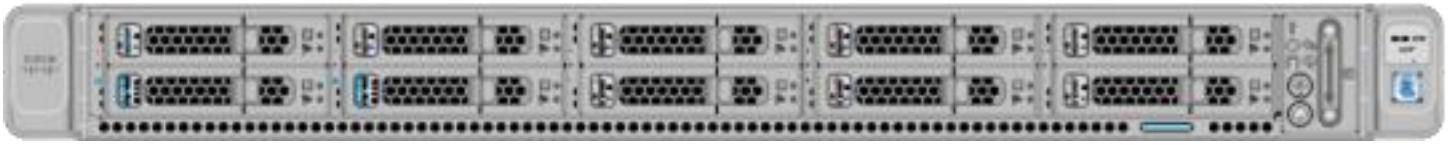
For more information, see: https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-b-series-blade-servers/datasheet-c78-739296.html.

Cisco UCS C220M5 Rack-Mount Server

The Cisco UCS C220 M5 Rack Server is among the most versatile general-purpose enterprise infrastructure and application servers in the industry. It is a high-density 2-socket rack server that delivers industry-leading performance and efficiency for a wide range of workloads, including virtualization, collaboration, and bare metal applications. The Cisco UCS C-Series Rack Servers can be deployed as standalone servers or as part of the Cisco Unified Computing System™ (Cisco UCS) to take advantage of Cisco's standards-based unified computing innovations that help reduce customers' Total Cost of Ownership (TCO) and increase their business agility. The Cisco UCS C220 M5 server extends the capabilities of the Cisco UCS portfolio in a 1-Rack-Unit (1RU) form factor. It incorporates the Intel® Xeon® Scalable processors, supporting up to 20 percent more cores per socket, twice the memory capacity, 20 percent greater storage density, and five times more PCIe NVMe Solid-State Disks (SSDs) compared to the previous generation of servers. These improvements deliver significant performance and efficiency gains that will improve your application performance. The C220 M5 delivers outstanding levels of expandability and performance in a compact package, with:

- Latest Intel Xeon Scalable CPUs with up to 28 cores per socket
- Up to 24 DDR4 DIMMs for improved performance
- Up to 10 Small-Form-Factor (SFF) 2.5-inch drives or 4 Large-Form-Factor (LFF) 3.5-inch drives (77 TB storage capacity with all NVMe PCIe SSDs)
- Support for 12-Gbps SAS modular RAID controller in a dedicated slot, leaving the remaining PCIe Generation 3.0 slots available for other expansion cards
- Modular LAN-On-Motherboard (mLOM) slot that can be used to install a Cisco UCS Virtual Interface Card (VIC) without consuming a PCIe slot
- Dual embedded Intel x550 10GBASE-T LAN-On-Motherboard (LOM) ports

Figure 3 Cisco UCS C220 M5SX



For more information, see: https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-c-series-rack-servers/datasheet-c78-739281.html.

Cisco UCS C240M5 Rack-Mount Server

The Cisco UCS C240 M5 Rack Server is a 2-socket, 2-Rack-Unit (2RU) rack server offering industry-leading performance and expandability. It supports a wide range of storage and I/O-intensive infrastructure workloads, from big data and analytics to collaboration. Cisco UCS C-Series Rack Servers can be deployed as standalone servers or as part of a Cisco Unified Computing System™ (Cisco UCS) managed environment to take advantage of Cisco's standards-based unified computing innovations that help reduce customers' Total Cost of Ownership (TCO) and increase their business agility.

In response to ever-increasing computing and data-intensive real-time workloads, the enterprise-class Cisco UCS C240 M5 server extends the capabilities of the Cisco UCS portfolio in a 2RU form factor. It incorporates the

Intel® Xeon® Scalable processors, supporting up to 20 percent more cores per socket, twice the memory capacity, and five times more

Non-Volatile Memory Express (NVMe) PCI Express (PCIe) Solid-State Disks (SSDs) compared to the previous generation of servers. These improvements deliver significant performance and efficiency gains that will improve your application performance. The C240 M5 delivers outstanding levels of storage expandability with exceptional performance, with:

- Latest Intel Xeon Scalable CPUs with up to 28 cores per socket
- Up to 24 DDR4 DIMMs for improved performance
- Up to 26 hot-swappable Small-Form-Factor (SFF) 2.5-inch drives, including 2 rear hot-swappable SFF drives (up to 10 support NVMe PCIe SSDs on the NVMe-optimized chassis version), or 12 Large-Form-Factor (LFF) 3.5-inch drives plus 2 rear hot-swappable SFF drives
- Support for 12-Gbps SAS modular RAID controller in a dedicated slot, leaving the remaining PCIe Generation 3.0 slots available for other expansion cards
- Modular LAN-On-Motherboard (mLOM) slot that can be used to install a Cisco UCS Virtual Interface Card (VIC) without consuming a PCIe slot, supporting dual 10- or 40-Gbps network connectivity
- Dual embedded Intel x550 10GBASE-T LAN-On-Motherboard (LOM) ports
- Modular M.2 or Secure Digital (SD) cards that can be used for boot

Figure 4 Cisco UCS C240 M5SX

For more information, see: https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-c-series-rack-servers/datasheet-c78-739279.html.

Cisco VIC Interface Cards

The Cisco UCS Virtual Interface Card (VIC) 1340 is a 2-port 40-Gbps Ethernet or dual 4 x 10-Gbps Ethernet, Fiber Channel over Ethernet (FCoE) capable modular LAN on motherboard (mLOM) designed exclusively for the M4 generation of Cisco UCS B-Series Blade Servers. All the blade servers for both Controllers and Computes will have MLOM VIC 1340 card. Each blade will have a capacity of 40Gb of network traffic. The underlying network interfaces like will share this MLOM card.

The Cisco UCS VIC 1340 enables a policy-based, stateless, agile server infrastructure that can present over 256 PCIe standards-compliant interfaces to the host that can be dynamically configured as either network interface cards (NICs) or host bus adapters (HBAs).

For more information, see: http://www.cisco.com/c/en/us/products/interfaces-modules/ucs-virtual-interface-card-1340/index.html.

The Cisco UCS Virtual Interface Card 1385 improves flexibility, performance, and bandwidth for Cisco UCS C-Series Rack Servers. It offers dual-port Enhanced Quad Small Form-Factor Pluggable (QSFP+) 40 Gigabit

Ethernet and Fibre Channel over Ethernet (FCoE) in a half-height PCI Express (PCIe) adapter. The 1385 card works with Cisco Nexus 40 Gigabit Ethernet (GE) and 10 GE switches for high-performance applications. The Cisco VIC 1385 implements the Cisco Data Center Virtual Machine Fabric Extender (VM-FEX), which unifies virtual and physical networking into a single infrastructure. The extender provides virtual-machine visibility from the physical network and a consistent network operations model for physical and virtual servers.

For more information, see: https://www.cisco.com/c/en/us/products/interfaces-modules/ucs-virtual-interface-card-1385/index.html.

Cisco UCS Fabric Extenders

Cisco UCS 2304 Fabric Extender brings the unified fabric into the blade server enclosure, providing multiple 40 Gigabit Ethernet connections between blade servers and the fabric interconnect, simplifying diagnostics, cabling, and management. It is a third-generation I/O Module (IOM) that shares the same form factor as the second-generation Cisco UCS 2200 Series Fabric Extenders and is backward compatible with the shipping Cisco UCS 5108 Blade Server Chassis. The Cisco UCS 2304 connects the I/O fabric between the Cisco UCS 6300 Series Fabric Interconnects and the Cisco UCS 5100 Series Blade Server Chassis, enabling a lossless and deterministic Fibre Channel over Ethernet (FCoE) fabric to connect all blades and chassis together. Because the fabric extender is similar to a distributed line card, it does not perform any switching and is managed as an extension of the fabric interconnects. This approach reduces the overall infrastructure complexity and enabling Cisco UCS to scale to many chassis without multiplying the number of switches needed, reducing TCO and allowing all chassis to be managed as a single, highly available management domain.

The Cisco UCS 2304 Fabric Extender has four 40Gigabit Ethernet, FCoE-capable, Quad Small Form-Factor Pluggable (QSFP+) ports that connect the blade chassis to the fabric interconnect. Each Cisco UCS 2304 can provide one 40 Gigabit Ethernet ports connected through the midplane to each half-width slot in the chassis, giving it a total eight 40G interfaces to the compute. Typically configured in pairs for redundancy, two fabric extenders provide up to 320 Gbps of I/O to the chassis.

Figure 5 Cisco UCS 2304 Fabric Extender

For more information, see: https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-6300-series-fabric-interconnects/datasheet-c78-675243.html.

Cisco Nexus 9000 Switches

The Cisco Nexus 9000 Series delivers proven high performance and density, low latency, and exceptional power efficiency in a broad range of compact form factors. Operating in Cisco NX-OS Software mode or in Application Centric Infrastructure (ACI) mode, these switches are ideal for traditional or fully automated data center deployments.

The Cisco Nexus 9000 Series Switches offer both modular and fixed 10/40/100 Gigabit Ethernet switch configurations with scalability up to 30 Tbps of non-blocking performance with less than five-microsecond latency, 1152 x 10 Gbps or 288 x 40 Gbps non-blocking Layer 2 and Layer 3 Ethernet ports and wire speed VXLAN gateway, bridging, and routing.

Figure 6 Cisco UCS Nexus 9396PX

For more information, see: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-736967.html.

Intel Scalable Processor Family

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

- Intel® Xeon® Scalable processors are now available in four feature configurations:
- Intel® Xeon® Bronze Processors with affordable performance for small business and basic storage.
- Intel® Xeon® Silver Processors with essential performance and power efficiency
- Intel® Xeon® Gold Processors with workload-optimized performance, advanced reliability
- Intel® Xeon® Platinum Processors for demanding, mission-critical AI, analytics, hybrid-cloud workloads

Figure 7 Intel® Xeon® Scalable Processor Family

Intel® SSD DC S4500 Series

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

Figure 8 Intel® SSD DC S4500

Red Hat OpenShift Container Platform

OpenShift Container Platform is Red Hat's container application platform that brings together Docker and Kubernetes and provides an API to manage these services. OpenShift Container Platform allows you to create and manage containers. Containers are standalone processes that run within their own environment, independent of operating system and the underlying infrastructure. OpenShift helps developing, deploying, and managing container-based applications. It provides a self-service platform to create, modify, and deploy applications on demand, thus enabling faster development and release life cycles. OpenShift Container Platform has a microservices-based architecture of smaller, decoupled units that work together. It runs on top of a Kubernetes cluster, with data about the objects stored in etcd, a reliable clustered key-value store.

Kubernetes Infrastructure

Within OpenShift Container Platform, Kubernetes manages containerized applications across a set of Docker runtime hosts and provides mechanisms for deployment, maintenance, and application-scaling. The Docker service packages, instantiates, and runs containerized applications.

A Kubernetes cluster consists of one or more masters and a set of nodes. This solution design includes HA functionality at the hardware as well as the software stack. Kubernetes cluster is designed to run in HA mode with 3 master nodes and 2 Infra nodes to help ensure that the cluster has no single point of failure.

Red Hat OpenShift Integrated Container Registry

OpenShift Container Platform provides an integrated container registry called OpenShift Container Registry (OCR) that adds the ability to automatically provision new image repositories on demand. This provides users with a built-in location for their application builds to push the resulting images. Whenever a new image is pushed to OCR, the registry notifies OpenShift Container Platform about the new image, passing along all the information about it, such as the namespace, name, and image metadata. Different pieces of OpenShift Container Platform react to new images, creating new builds and deployments.

Container-native Storage Solution from Red Hat

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

Container-native storage for OpenShift Container Platform is built around three key technologies:

- OpenShift provides the Platform-as-a-Service (PaaS) infrastructure based on Kubernetes container management. Basic OpenShift architecture is built around multiple master systems where each system contains a set of nodes.
- Red Hat Gluster Storage provides the containerized distributed storage based on Red Hat Gluster Storage container. Each Red Hat Gluster Storage volume is composed of a collection of bricks, where each brick is the combination of a node and an export directory.
- Heketi provides the Red Hat Gluster Storage volume life cycle management. It creates the Red Hat Gluster Storage volumes dynamically and supports multiple Red Hat Gluster Storage clusters.

Docker

Red Hat OpenShift Container Platform uses Docker runtime engine for containers.

Kubernetes

Red Hat OpenShift Container Platform is a complete container application platform that natively integrates technologies like Docker and Kubernetes; a powerful container cluster management and orchestration system.

Etcd

Etcd is a key-value store used in OpenShift Container Platform cluster. Etcd data store provides complete cluster and endpoint states to the OpenShift API servers. Etcd data store furnishes information to API servers about node status, network configurations, secrets, etc.

Open vSwitch

Open vSwitch is an open-source implementation of a distributed virtual multilayer switch. It is designed to enable effective network automation through programmatic extensions, while supporting standard management interfaces and protocols such as 802.1ag, SPAN, LACP, and NetFlow. Open vSwitch provides software-defined networking (SDN)-specific functions in the OpenShift Container Platform environment.

HAProxy

HAProxy is open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers. In this solution, HAProxy is deployed in virtual machine in VMware HA cluster which provides routing and load-balancing functions for Red Hat OpenShift applications. Other instance of HAProxy acts as an ingress router for all applications deployed in Red Hat OpenShift cluster.

The external load balancer can also be utilized. However, the configuration of the external load balancer is out of the scope of this document.

Red Hat Ansible Automation

Red Hat Ansible Automation is a powerful IT automation tool. It is capable of provisioning numerous types of resources and deploying applications. It can configure and manage devices and operating system components. Due to the simplicity, extensibility, and portability, this OpenShift solution is based largely on Ansible Playbooks.

Ansible is mainly used for installation and management of the Red Hat OpenShift Container Platform deployment.

For VMware environment, the installation of Red Hat OpenShift Container Platform is done via the Ansible playbooks installed by the openshift-ansible-playbooks rpm package. In order to deploy 3 masters, 3 infrastructure, and 3 app nodes, Ansible playbooks are utilized. These playbooks can be altered for VM sizing and to meet other specific requirements if needed. Details of these playbooks can be obtained from the following https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/vmware-ansible/playbooks.

Solution Design

This section provides an overview of the hardware and software components used in this solution, as well as the design factors to be considered in order to make the system work as a single, highly available solution.

Hardware and Software Revisions

IMPORTANT The following hardware and software versions have been validated in Red Hat OpenShift Container Platform 3.9.33.

Table 1 lists the firmware versions validated in this RHOCP solution.

Table 1 **Hardware Revisions**

Hardware	Firmware Versions
Cisco UCS Manager	3.2.3d
Cisco UCS B200 M5 Server	3.2.3d
Cisco UCS Fabric Interconnects 6332UP	3.2.3d

For information about the OS version and system type, see Cisco Hardware and Software Compatibility.

Table 2 lists the software versions validated in this RHOCP solution.

Table 2 **Software Versions**

Software	Versions
Red Hat Enterprise Linux	7.5
Red Hat OpenShift Container Platform	3.9.33
VMware vSphere	6.7
Kubernetes	1.9
Docker	1.13.1
Red Hat Ansible Engine	2.4.6.0
Etcd	3.2.22
Open vSwitch	2.9.0
Red Hat Gluster Storage	3.3.0
Gluster FS	3.8.4

Solution Components

This solution is validated comprised of following components (see Table 3).

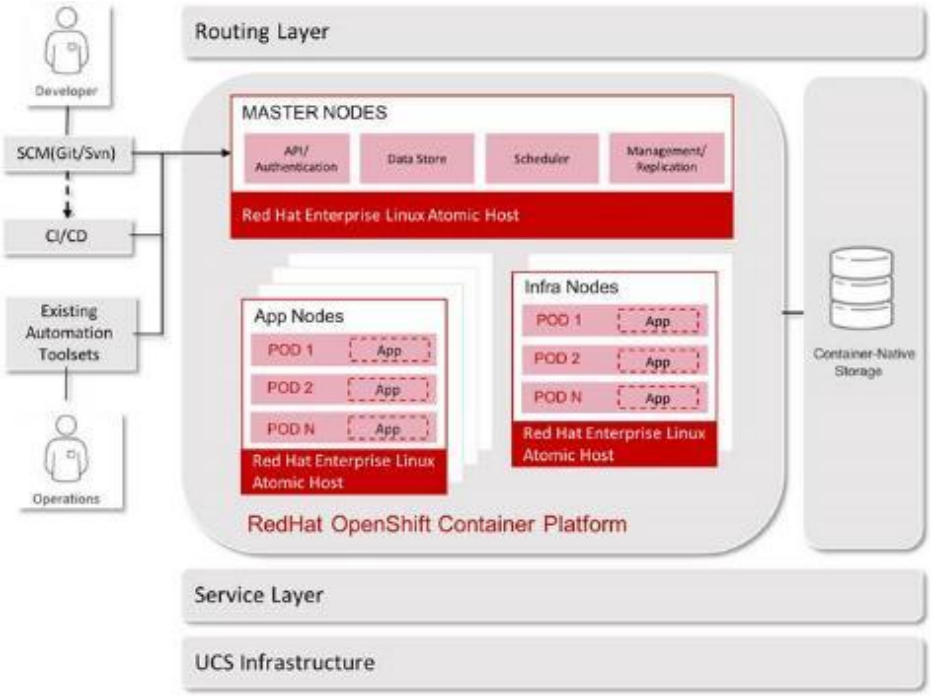
Table 3 **Solution Components**

Component	Model	Quantity
ESXi hosts for master, infra, and application VMs	Cisco UCS B200 M5 Servers	4
Storage Nodes	Cisco UCS C240M5SX	3
Chassis	Cisco UCS 5108 Chassis	1
IO Modules	Cisco UCS 2304XP Fabric Extenders	2
Fabric Interconnects	Cisco UCS 6332 Fabric Interconnects	2
Nexus Switches	Cisco Nexus 93180YC-EX Switches	2

Architectural Overview

Red Hat OpenShift Container Platform is managed by the Kubernetes container orchestrator, which manages containerized applications across a cluster of systems running the Docker container runtime. The physical configuration of Red Hat OpenShift Container Platform is based on the Kubernetes cluster architecture. OpenShift is a layered system designed to expose underlying Docker-formatted container image and Kubernetes concepts as accurately as possible, with a focus on easy composition of applications by a developer. For example, install Ruby, push code, and add MySQL. The concept of an application as a separate object is removed in favor of more flexible composition of "services", allowing two web containers to reuse a database or expose a database directly to the edge of the network.

Figure 9 Architectural Overview



This Red Hat OpenShift Container Platform reference architecture contains five types of nodes: bastion, master, infrastructure, storage, and application.

Bastion Node

This is a dedicated node that serves as the main deployment and management server for the Red Hat OpenShift cluster. It is used as the logon node for the cluster administrators to perform the system deployment and management operations, such as running the Ansible OpenShift deployment Playbooks and performing scale-out operations. Also, Bastion node runs DNS services for the OpenShift Cluster nodes. The bastion node runs Red Hat Enterprise Linux 7.5.

OpenShift Master Nodes

The OpenShift Container Platform master is a server that performs control functions for the whole cluster environment. It is responsible for the creation, scheduling, and management of all objects specific to Red Hat OpenShift. It includes API, controller manager, and scheduler capabilities in one OpenShift binary. It is also a common practice to install an etcd key-value store on OpenShift masters to achieve a low-latency link between etcd and OpenShift masters. It is recommended that you run both Red Hat OpenShift masters and etcd in highly available environments. This can be achieved by running multiple OpenShift masters in conjunction with an external active-passive load balancer and the clustering functions of etcd. The OpenShift master node runs Red Hat Enterprise Linux Atomic Host 7.5.

OpenShift Infrastructure Nodes

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

Additional applications and services can be deployed on OpenShift infrastructure nodes. The OpenShift infrastructure node runs Red Hat Enterprise Linux Atomic Host 7.5.

OpenShift Application Nodes

The OpenShift application nodes run containerized applications created and deployed by developers. An OpenShift application node contains the OpenShift node components combined into a single binary, which can be used by OpenShift masters to schedule and control containers. A Red Hat OpenShift application node runs Red Hat Enterprise Linux Atomic Host 7.5.

OpenShift Storage Nodes

The OpenShift storage nodes run containerized GlusterFS services which configure persistent volumes for application containers that require data persistence. Persistent volumes may be created manually by a cluster administrator or automatically by storage class objects. An OpenShift storage node is also capable of running containerized applications. A Red Hat OpenShift storage node runs Red Hat Enterprise Linux Atomic Host 7.5.

Table 4 lists the functions and roles for each class of node in this solution for the OpenShift Container Platform.

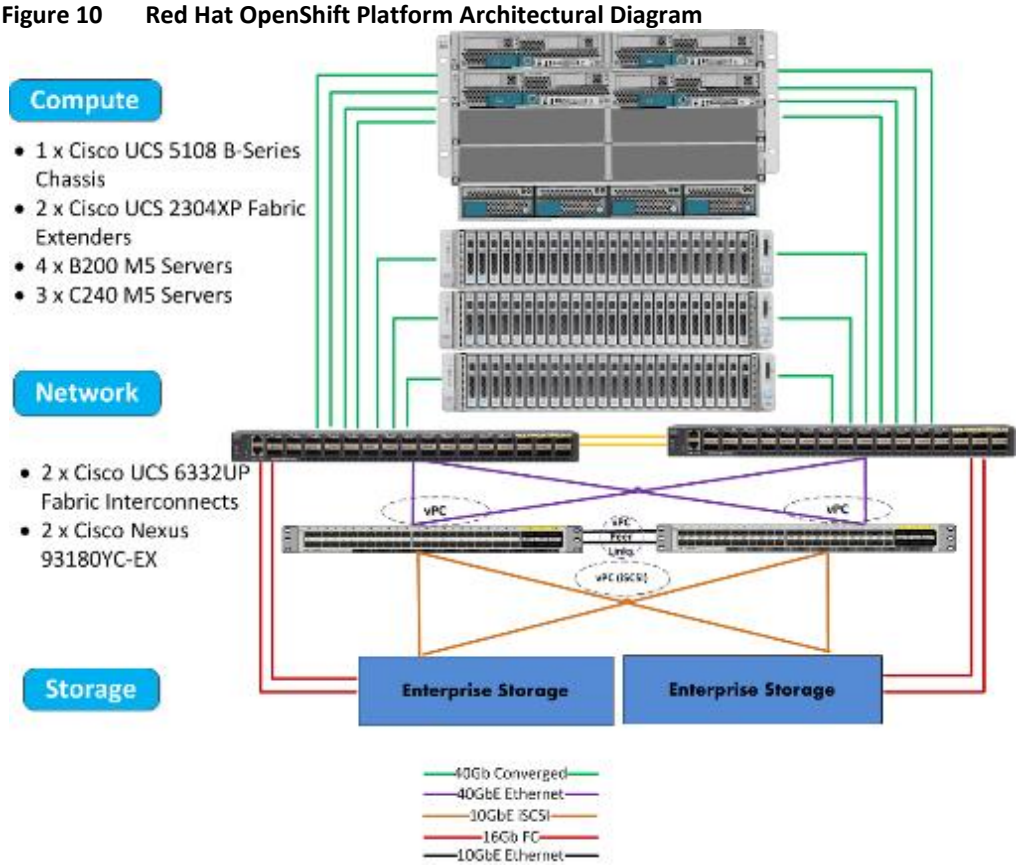
Table 4 **Type of Nodes in OpenShift Container Platform Cluster and their Roles**

Node	Roles
Bastion Node	- System deployment and Management Operations - Runs Ansible playbooks. - It can also be configured as IP Router that routes traffic across all the nodes via the control network
Master Nodes	- Kubernetes services - Etcd data store - Controller Manager & Scheduler - API services
Infrastructure Nodes	- Container Registry - Heketi - HA Proxy Router
Application Nodes	- Application Containers PODs - Docker Runtime
Storage Nodes	- Red Hat Gluster Storage - Container-native storage services - Storage nodes are labeled `compute`, so workload scheduling is enabled by default

Physical Topology

Figure 10 shows the physical architecture used in this reference design.

Figure 10 Red Hat OpenShift Platform Architectural Diagram



3 x C240s are acting as compute nodes as well as providing persistent storage to containers.

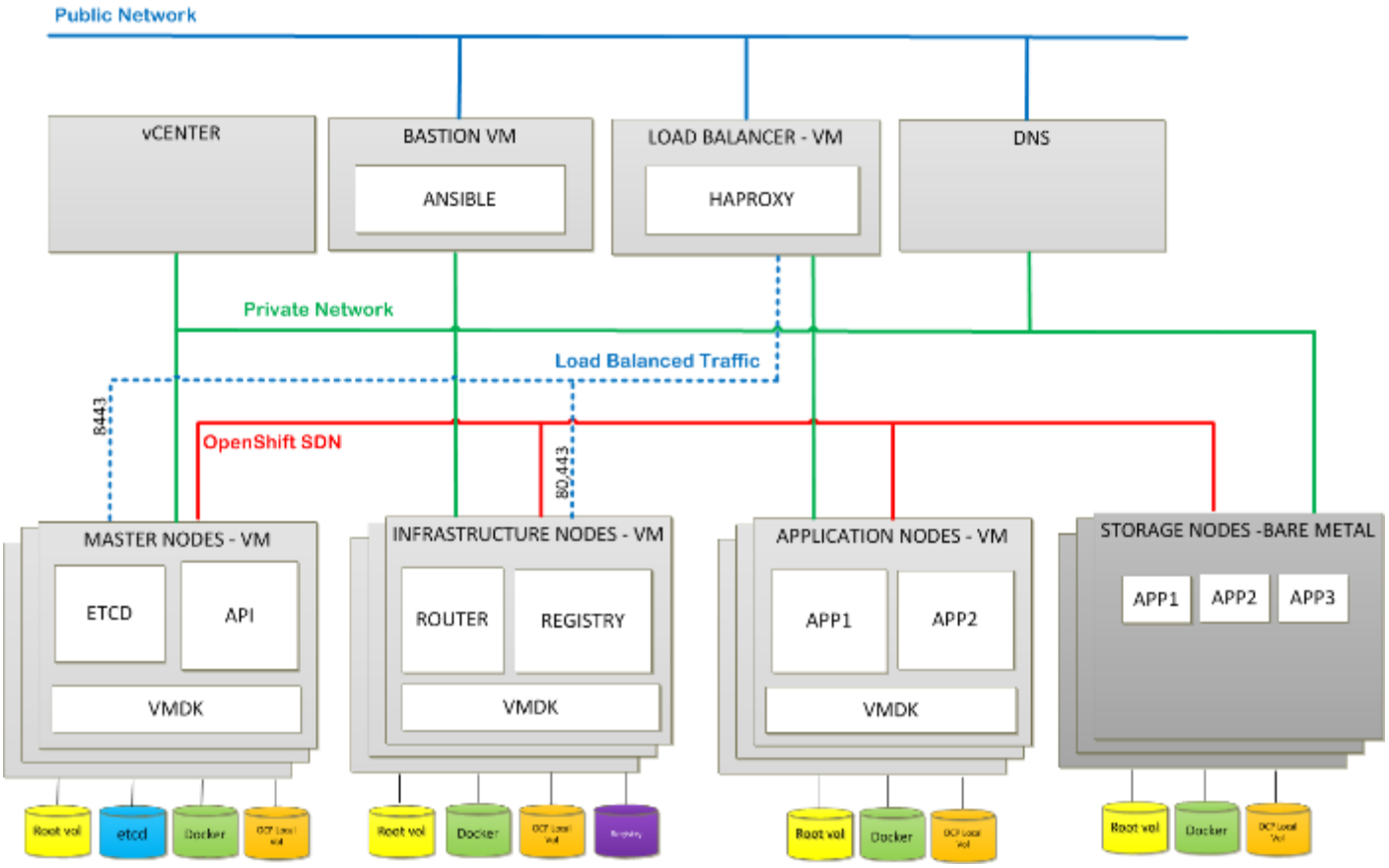
Enterprise storage is providing Boot LUNs to all the servers via the boot policy configuration in Cisco UCS Manager. However, Cisco UCS C240s can also be configured to boot from local disk by configuring storage profiles in Cisco UCS Manager with RAID 1 Mirrored.

Enterprise storage also provides data stores to VMware vSphere environment for VM virtual disks.

Logical Topology

Figure 11 illustrates the logical topology of Red Hat OpenShift Container Platform.

Figure 11 Logical Diagram



- Private Network – This network is common to all nodes. This is internal network and mostly used by bastion node to perform SSH access for Ansible playbook. Hence, bastion node is acting as a jump host for SSH. Outbound NAT has been configured for nodes to access Red Hat content delivery network for enabling the required repos.
- Bastion node is a part of public subnet and it can also be configured as IP Router that routes traffic across all the nodes via the control network. In this case default gateway of all the nodes will be bastion node private IP. This scenario would limit outside access for all the nodes if bastion node is powered down
- OpenShift SDN – OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between pods across the OpenShift Container Platform cluster. This pod network is established and maintained by the OpenShift SDN, which configures an overlay network using Open vSwitch (OVS).
- Public Network – This network provides internet access.

Virtual Machine Instance Details

Table 5 lists the minimum VM requirements for each node.

Table 5 **VM Configurations**

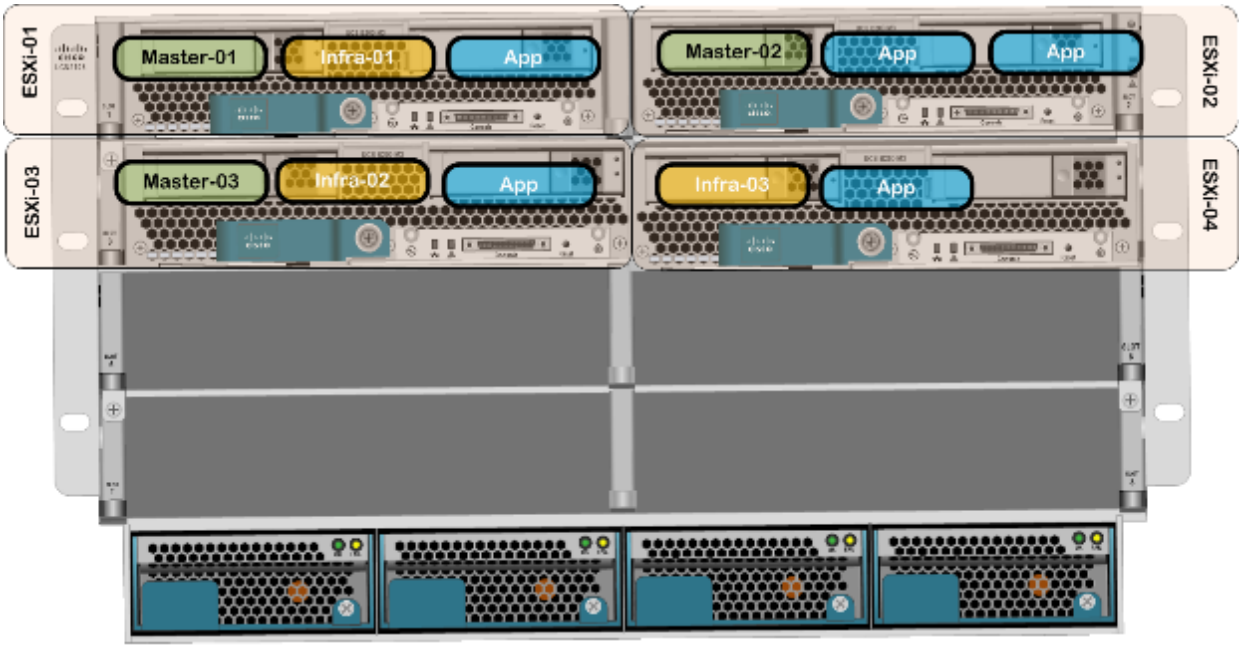
Node Type	CPUs	Memory	Disk 1	Disk 2	Disk 3	Disk 4
Master	2 vCPU	16GB RAM	1 x 60GB - OS RHEL 7.5	1 x 40GB - Docker volume	1 x 40Gb - EmptyDir volume	1 x 40GB - ETCD volume
Infra Nodes	2 vCPU	8GB RAM	1 x 60GB - OS RHEL 7.5	1 x 40GB - Docker volume	1 x 40Gb - EmptyDir volume	1 x 300GB for Registry
App Nodes	2 vCPU	8GB RAM	1 x 60GB - OS RHEL 7.5	1 x 40GB - Docker volume	1 x 40Gb - EmptyDir volume	
Bastion	1 vCPU	4GB RAM	1 x 60GB - OS RHEL 7.5			

Master nodes should contain three extra disks used for Docker storage, etcd, and OpenShift volumes for OCP pod storage. All volumes are thin provisioned. Infra and App nodes does not require etcd. In this reference architecture, VMs are deployed in a single cluster in a single datacenter.

When planning an environment with multiple masters, a minimum of three etcd hosts and one load-balancer between the master hosts are required.

Red Hat OpenShift Container Platform Node Placement

The following diagram shows how the master, infrastructure, and application nodes should be placed in vSphere ESXi hosts.



- It is recommended to configure anti-affinity rule for master VMs after they are installed from Ansible play-book. Setting up the anti-affinity rule is described in a subsequent section of this document.
- Within vSphere environment, HA is maintained by VMware vSphere High-availability

HA Proxy Load Balancer

In the reference architecture, HA Proxy load balancer is used. However, on premise existing load balancer can also be utilized. HA Proxy is the entry point for many Red Hat OpenShift Container Platform components. OpenShift Container Platform console is accessible via the master nodes, which is spread across multiple instances to provide load balancing as well as high availability.

Application traffic passes through the Red Hat OpenShift Container Platform Router on its way to the container processes. The Red Hat OpenShift Container Platform Router is a reverse proxy service container that multiplexes the traffic to multiple containers making up a scaled application running inside Red Hat OpenShift Container Platform. The load balancer used by *infra* nodes acts as the public view for the Red Hat OpenShift Container Platform applications.

The destination for the master and application traffic must be set in the load balancer configuration after each instance is created, the floating IP address is assigned and before the installation. A single `haproxy` load balancer can forward both sets of traffic to different destinations.

Deployment Hardware and Software

Solution Prerequisites

The following prerequisites must be met before starting the deployment of Red Hat OpenShift Container platform:

- An active Red Hat account with access to the OpenShift Container Platform subscriptions through purchased entitlements.
- Fully functional DNS server is an absolute MUST for this solution. Existing DNS server can be utilized as long as it is accessible to and from OpenShift nodes and can provide name resolutions to hosts and containers running on the platform.
- OpenShift Enterprise requires NTP to synchronize the system and hardware clocks. It prevents master and nodes in the cluster from going out of sync.
- Network Manager is required on the nodes for populating dnsmasq with the DNS IP addresses.
- Have the Red Hat Enterprise Linux Server 7.5 ISO image (rhel-server-7.5-x86-64-dvd.iso) on hand and readily available.
- Pre-existing VMware vSphere and vCenter environment with the capability of vCenter High Availability.

Required Channels

Subscription to the following channels are required. Before you start the deployment, make sure your subscriptions have access to these channels.

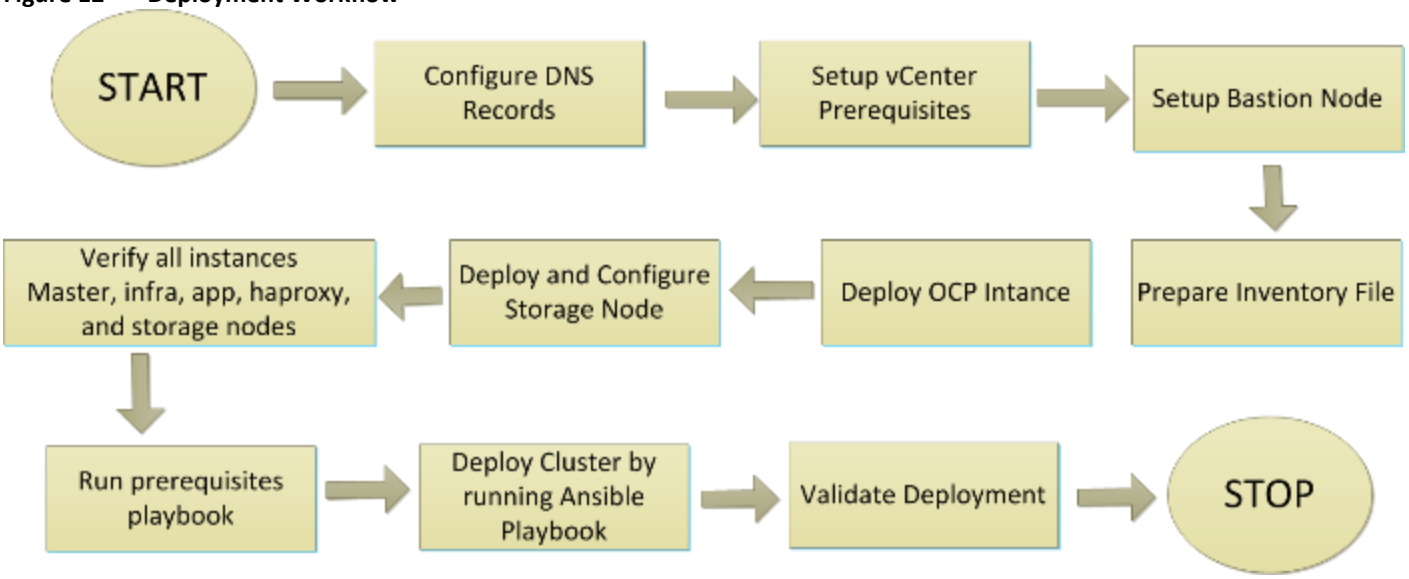
Table 6 **Red Hat OpenShift Container Platform Required Channels**

Channel	Repository Name
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms
Red Hat OpenShift Container Platform 3.9 (RPMs)	rhel-7-server-ose-3.9-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms
Red Hat Enterprise Linux Fast Datapath (RHEL 7 Server) (RPMs)	rhel-7-fast-datapath-rpms
Red Hat Ansible Engine 2.4 RPMs for Red Hat Enterprise Linux 7 Server	rhel-7-server-ansible-2.4-rpms

Deployment Workflow

Figure 12 shows the workflow of deployment process.

Figure 12 Deployment Workflow



DNS (Domain Name Server) Configuration

DNS is a mandatory requirement for successful Red Hat OpenShift Container Platform deployment. It should provide name resolution to all the hosts and containers running on the platform.

It is very important to note that adding entries into /etc/hosts file on each host cannot replace the requirement of DNS server as this file is not copied into the containers running on the platform. If you do not have the functional DNS server, installation will fail.

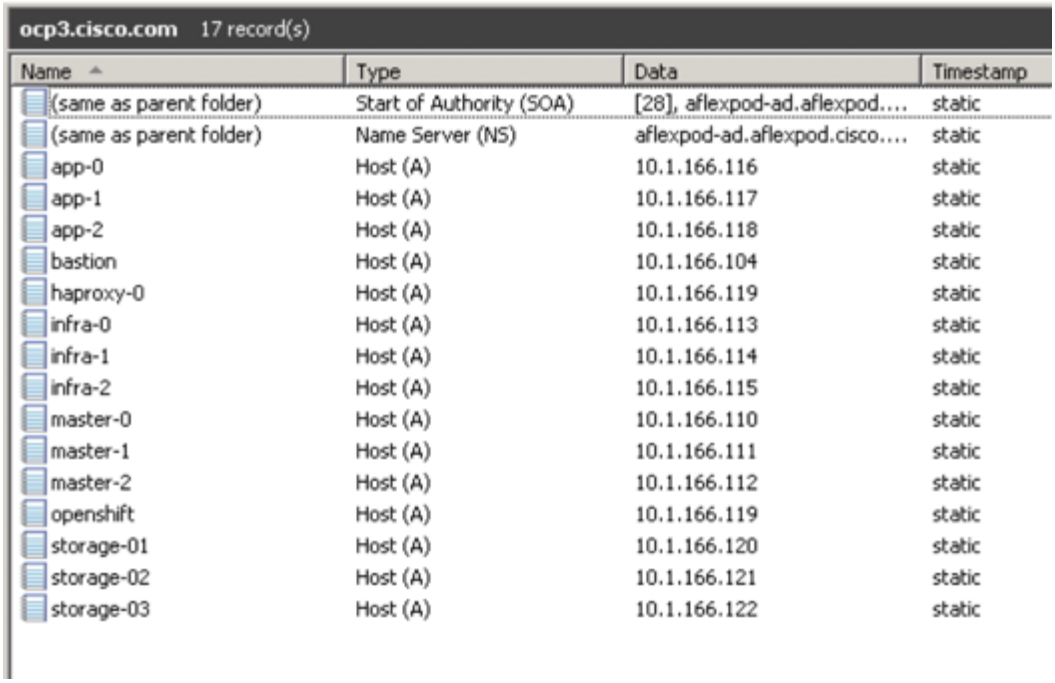
Below is the sample DNS configuration for reference:

```
$ORIGIN apps.ocp3.cisco.com
*   A       192.x.x.240
$ORIGIN ocp3.cisco.com.
haproxy-0   A 10.x.y.200
infra-0     A 10.x.y.100
infra-1     A 10.x.y.101
infra-2     A 10.x.y.102
master-0    A 10.x.y.103
master-1    A 10.x.y.104
master-2    A 10.x.y.105
app-0       A 10.x.y.106
app-1       A 10.x.y.107
app-2       A 10.x.y.108
storage-01  A 10.x.y.109
storage-02  A 10.x.y.110
storage-03  A 10.x.y.111
openshift   A 10.x.x.200
$ORIGIN external.ocp3.cisco.com
openshift   A 192.x.x.240
bastion     A 192.x.x.120
```

Setting up DNS server is beyond the scope of this document. For this reference architecture, it is assumed that functional DNS already exists.

Figure 13 shows forward lookup zone ocp3.cisco.com and the DNS records used to point to sub-domain to an IP address.

Figure 13 DNS Configuration



ocp3.cisco.com 17 record(s)

Name	Type	Data	Timestamp
(same as parent folder)	Start of Authority (SOA)	[28], aflexpod-ad.aflexpod....	static
(same as parent folder)	Name Server (NS)	aflexpod-ad.aflexpod.cisco....	static
app-0	Host (A)	10.1.166.116	static
app-1	Host (A)	10.1.166.117	static
app-2	Host (A)	10.1.166.118	static
bastion	Host (A)	10.1.166.104	static
haproxy-0	Host (A)	10.1.166.119	static
infra-0	Host (A)	10.1.166.113	static
infra-1	Host (A)	10.1.166.114	static
infra-2	Host (A)	10.1.166.115	static
master-0	Host (A)	10.1.166.110	static
master-1	Host (A)	10.1.166.111	static
master-2	Host (A)	10.1.166.112	static
openshift	Host (A)	10.1.166.119	static
storage-01	Host (A)	10.1.166.120	static
storage-02	Host (A)	10.1.166.121	static
storage-03	Host (A)	10.1.166.122	static

The name of the Red Hat OpenShift Container Platform console is the address of the haproxy-0 instance on the network. For example, openshift.ocp3.cisco.com DNS name has the IP address of ha-proxy node.

Application DNS

Applications served by OpenShift are accessible by the router on ports 80/TCP and 443/TCP. The router uses a wildcard record to map all host names under a specific sub domain to the same IP address without requiring a separate record for each name. This allows Red Hat OpenShift Container Platform to add applications with arbitrary names as long as they are under that sub domain.

For example, a wildcard record for *.apps.ocp3.cisco.com causes DNS name lookups for wordpress.apps.ocp3.cisco.com and nginx-01.apps.ocp3.cisco.com to both return the same IP address: 192.168.91.240. All traffic is forwarded to the OpenShift Routers. The Routers examine the HTTP headers of the queries and forward them to the correct destination.

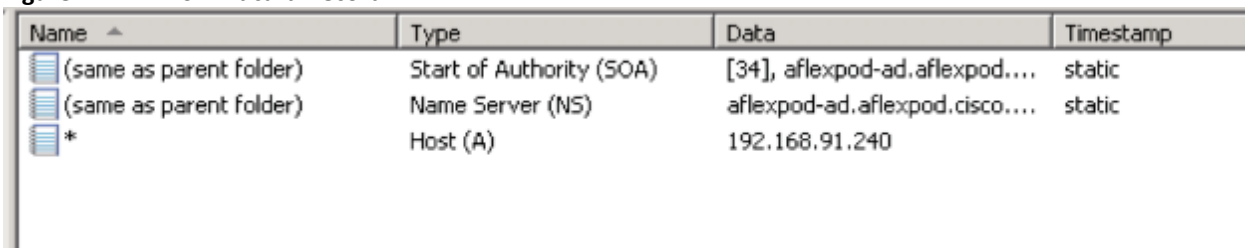
With a load-balancer host address of 192.168.91.240, the wildcard DNS record can be added as follows:

Table 7 **Load Balancer DNS records**

IP Address	Host Name	Purpose
192.168.91.240	*.apps.ocp3.cisco.com	User access to application webs services.

Figure 14 shows wildcard DNS record that resolves to the IP address of the OpenShift router.

Figure 14 DNS Wildcard Record



VMware vCenter Prerequisites

This reference architecture assumes a pre-existing VMware vCenter environment and is configured based on the best practices for the infrastructure.

VMware HA and storage IO control should already be configured. Once the environment is setup, anti-affinity rules are recommended to be setup for maximum uptime and optimal performance.

Networking

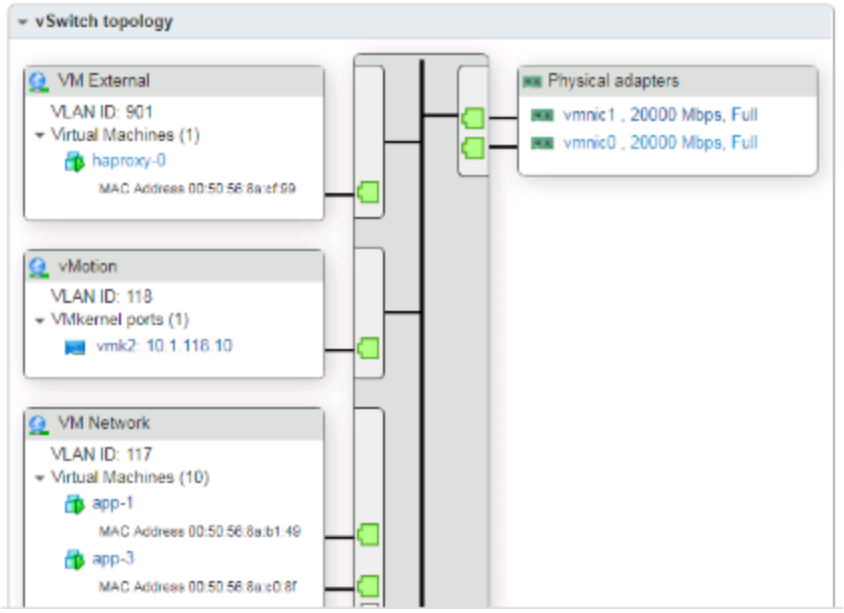
An existing port group and virtual LAN (VLAN) are required for deployment. The initial configuration of the Red Hat OpenShift nodes in this reference architecture assumes you will be deploying VMs to a port group called "VM Network".

The environment can utilize a VMware vSphere Distributed Switch (VDS) or Standard vSwitch. The specifics of that are unimportant and beyond the scope of this document. However, a vDS is required if you wish to utilize network IO control and some of the quality of service (QoS) technologies.

Furthermore, specifics of setting up the VMware environment on Cisco UCS is not covered in this document such as physical NICs, storage configuration whether IP based or Fibre channel, UCS vNIC failover and redundancy, vNIC templates, service profiles, other policies, and so on. These design choices and scenarios have been previously validated in various Cisco Validated Designs. For best practices and supported design, see:

https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/flexpod_esxi65u1_n9kiscsi.html?referring_site=RE&pos=3&page=https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/flexpod_esxi65u1design.html

The figure below shows the standard vSwitch configuration used in this reference architecture. After the VM deployment is completed from Ansible playbook (described in a subsequent section), vSphere vDS can be configured, details of which are outlined in the link above.



vCenter Shared Storage

All vSphere hosts should have shared storage to provision VMware virtual machine disk files (VMDKs) for templates. It is also recommended to enable storage I/O control (SIOC) to address any latency issues caused by performance. For in-depth overview of storage IO control, see: https://kb.vmware.com/s/article/1022091.

vSphere Parameter

Table 8 lists the vSphere parameter required for inventory file discussed in later section of this document. These parameter must be handy and pre-configured in vCenter environment before preparing the inventory file.

Table 8 vSphere Parameter

Parameter	Description	Example or Defaults
openshift_cloudprovider_vsphere_host	IP or Hostname of vCenter Server	
openshift_cloudprovider_vsphere_username	Username of vCenter Administrator	'administrator@vsphere.local'
openshift_cloudprovider_vsphere_password	Password of vCenter administrator	
openshift_cloudprovider_vsphere_cluster	Cluster to place VM in	OCP-Cluster
openshift_cloudprovider_vsphere_datacenter	Datacenter to place VM in	OCP-Datacenter
openshift_cloudprovider_vsphere_datastore	Datastore for VM VMDK	datastore4
openshift_cloudprovider_vsphere_resource_pool	Resource pool to be used for newly created VMs	ocp39

Parameter	Description	Example or Defaults
openshift_cloudprovider_vsphere_folder=	Folder to place newly created VMs in	ocp39
openshift_cloudprovider_vsphere_template	Template to clone new VM from	RHEL75
openshift_cloudprovider_vsphere_vm_network	Destination network for VMs. (vSwitch or VDS)	VM Network
openshift_cloudprovider_vsphere_vm_netmask	Network Mask for VM network	255.255.255.0
openshift_cloudprovider_vsphere_vm_gateway	Gateway of VM Network	10.1.166.1
openshift_cloudprovider_vsphere_vm_dns	DNS for VM	10.1.166.9

Resource Pool, Cluster Name, and Folder Location

This reference architecture assumes some default names as per the playbook used to deploy the cluster VMs. However, it is recommend to follow the information mentioned in Table 8.

Create the following per Table 8:

1. Create a resource pool named: "ocp3"
2. Create a folder for the Red Hat OpenShift VMs for logical organization named: "ocp"
3. Make sure this folder exists under the datacenter and cluster you will use for deployment

If you would like customize the names, remember to specify them later while creating the inventory file.

Prepare RHEL VM Template

This reference architecture is based on RHEL 7.5. VM template and needs to be prepared for use with the RHOCP instance deployment.

To prepare the RHEL VM template, complete the following steps:

1. Create a Virtual machine using RHEL 7.5 iso. You can create the VM with 2 vCPU, 4GB RAM, and 50GB disk for RHEL 7.5 OS. Specify net.ifnames=0 biosdevname=0 after pressing tab key as boot parameter to get consistent naming for network interfaces.
2. Power-On VM, configure /etc/sysconfig/network-script/ifcfg-eth0 with IP address, Netmask, Gateway, and DNS. Restart networking by running the following command.

```
# systemctl restart network
```

3. Register VM with Red Hat OpenShift subscription and enable required repositories:

```
subscription-manager register --username <username> --password <password>
subscription-manager list --available
subscription-manager attach --pool=<pool-id>
subscription-manager repos --disable='*'
subscription-manager repos --enable="rhel-7-server-rpms" --enable="rhel-7-server-extras-rpms" --
enable="rhel-7-server-ose-3.9-rpms" --enable="rhel-7-fast-datapath-rpms" --enable="rhel-7-server-ansible-
2.4-rpms"
```

4. Install following packages:

```
yum install -y open-vm-tools PyYAML perl net-tools chrony python-six iptables iptables-services
```

These packages are also installed by vmware-guest-setup TASK in ansible playbook for deploying production VM. However, if it is not pre-installed, playbook will not be able to configure instance IP address and hostname, which will fail subsequent steps in openshift-ansible-contrib/reference-architecture/vmware-ansible/playbooks/prod.yaml

5. Unregister the VM:

```
# subscription-manager unregister
# subscription-manager clean
```

6. Follow the steps mentioned in https://access.redhat.com/solutions/198693 to create clean VM for use as a template or cloning.
7. Change the /etc/sysconfig/network-script/ifcfg-eth0 with the following:

```
TYPE=Ethernet
BOOTPROTO=none
DEVICE=eth0
NAME=eth0
ONBOOT=yes
```

8. Run the following:

```
# ifdown eth0
```

9. Power-off the VM and convert it into template by Right click VM > Template > Convert to Template.

Setting Up Bastion Instance

The Bastion host serves as the installer of the Ansible playbooks that deploy Red Hat OpenShift Container Platform as well as an entry point for management tasks.

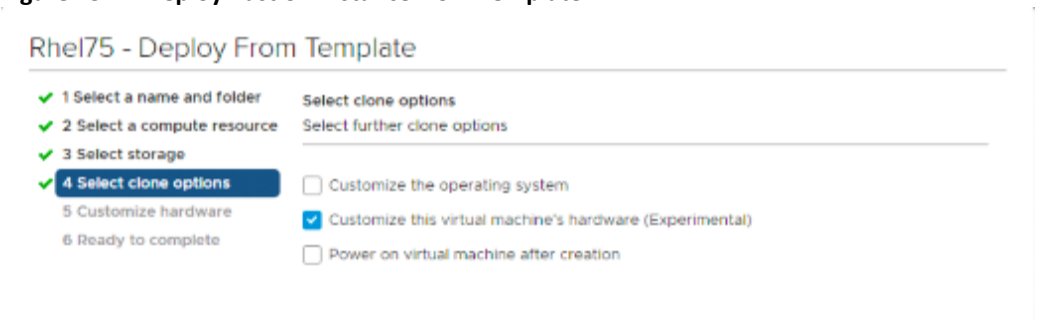
Bastion instance is a non-OpenShift instance accessible from outside of the Red Hat OpenShift Container Platform environment, configured to allow remote access via secure shell (ssh). To remotely access an instance, the systems administrator first accesses the bastion instance, then "jumps" via another ssh connection to the intended OpenShift instance. The bastion instance may be referred to as a "jump host".

As the bastion instance can access all internal instances, it is recommended to take extra measures to harden this instance's security. For more information about hardening the bastion instance, visit the following guide https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/index

To set up the bastion instance, complete the following steps:

1. Deploy a VM from Rhel75 template. Customize the virtual machine's hardware as shown below based on the requirements mentioned in Table 1.

Figure 15 Deploy Bastion Instance from Template



2. After the VM is deployed, configure eth0 interface as shown below:

```
[root@bastion ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=static
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.1.166.104
NETMASK=255.255.255.0
GATEWAY=10.1.166.1
DNS1=10.1.166.9
```

3. Set the host name:

```
[root@bastion ~]#hostnamectl set-hostname bastion.ocp3.cisco.com
[root@bastion ~]#hostnamectl status
```

4. Register Bastion instance with Red Hat OpenShift subscription and enable required repositories:

```
[root@bastion ~]# subscription-manager register --username <username> --password <password>
[root@bastion ~]# subscription-manager list --available
[root@bastion ~]# subscription-manager attach --pool=<pool-id>
[root@bastion ~]# subscription-manager repos --disable='*'
[root@bastion ~]# subscription-manager repos --enable="rhel-7-server-rpms" --enable="rhel-7-server-
extras-rpms" --enable="rhel-7-server-ose-3.9-rpms" --enable="rhel-7-fast-datapath-rpms" --enable="rhel-7-
server-ansible-2.4-rpms"
```

Red Hat subscription might fail if the system date is too far off with current date and time.

5. Install atomic-openshift-util package via following command:

```
[root@bastion ~]#sudo yum install -y ansible atomic-openshift-utils git
```

Configure Ansible

Ansible is installed on the deployment instance to perform the registration, installation of packages, and the deployment of the Red Hat OpenShift Container Platform environment on the master and node instances.

Before running playbooks, it is important to create an ansible.cfg to reflect the deployed environment:

```
 [root@dephost ansible]# cat ~/ansible.cfg
[defaults]
forks = 20
host_key_checking = False
roles_path = roles/
gathering = smart
remote_user = root
private_key = ~/.ssh/id_rsa
```

```
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
nocows = 1
callback_whitelist = profile_tasks
[ssh_connection]
ssh_args = -C -o ControlMaster=auto -o ControlPersist=900s -o GSSAPIAuthentication=no -o
PreferredAuthentications=publickey control_path = %(directory)s/%%h-%%r
pipelining = True
timeout = 10
[persistent_connection]
connect_timeout = 30
connect_retries = 30
connect_interval = 1
[root@dephost ansible]#
```

Prepare Inventory File

Ansible Playbook execution requires an inventory file to perform tasks on multiple systems at the same time. It does this by selecting portions of systems listed in Ansible's inventory, which defaults to being saved in the location /etc/ansible/hosts on the Bastion node for this solution. This inventory file consists of Hosts and Groups, Host Variables, Group Variables and behavioral Inventory Parameters.

This inventory file is customized for the Playbooks we used in this solution for preparing nodes and installing OpenShift Container Platform. The inventory file also describes the configuration of our OCP cluster and include/exclude additional OCP components and their configurations.

Table 9 table lists the sections used in the inventory file.

Table 9 **Inventory File Sections and their Descriptions**

Sections	Description
[OSEv3:children]	- This section defines set of target systems on which Playbook tasks will be executed - Target system groups used in this solution are – masters, nodes, etcd, lb, local and glusterfs - glusterfs host group are the storage nodes hosting CNS and Container Registry storage services
[OSEv3:vars]	- This section is used for defining OCP Cluster variables - These are environmental variable that are used during Ansible install and applied globally to the OCP cluster
[local]	- This host group points to the Bastion node - All tasks assigned for host group local gets applied to Bastion node only
[masters]	- All master nodes, master-0/1/2.ocp3.cisco.com are grouped under this section
[nodes]	- Host group contains definition of all nodes that are part of OCP cluster including master nodes - Both `containerized` and `openshift_schedulable` is set to true except for master nodes

Sections	Description
[etcd]	- Host group having nodes which will run etcd data store services - In this solution master node co-host etcd data store as well - Host names given are master-0/1/2.ocp3.cisco.com. These names will resolve to master node hosts
[lb]	- This host group is for nodes which will run load-balancer – OpenShift_loadbalancer/haproxy-router
[glusterfs]	- All storage nodes are grouped together under this - CNS services run on these nodes as glusterfs PODs - `glusterfs_devices` variable is used to allocate physical storage JBOD devices for container-native storage
[glusterfs_registry]	This will have entries for each node that will host glusterfs-backend registry and include glusterfs_devices.

Table 10 lists the inventory key variables used in this solution and their descriptions.

Table 10 **Inventory Key Variables Used in this Solution and Descriptions**

Key Variables	Description
deployment_type=openshift-enterprise openshift_release=v3.9 os_sdn_network_plugin_name ='redhat/openshift-ovs-multitenant'	- With this variable OpenShift Container Platform gets deployed - OCP release version as used in this solution - This variable configures which OpenShift SDN plug-in to use for the pod network, which defaults to redhat/openshift-ovs-subnet for the standard SDN plug-in. `redhat/openshift-ovs-multitenant` value enable multi-tenanc
openshift_hosted_manage_registry=true openshift_hosted_registry_storage_kind=glusterfs openshift_hosted_registry_storage_volume_size=30Gi	- These variables make Ansible Playbook to install OpenShift managed internal image registry - Storage definition for the image repository. In our Solution we use GlusterFS as back-end storage - Pre-provisioned Volume size for the repo.
openshift_master_dynamic_provisioning_enabled=True openshift_storage_glusterfs_storageclass=true openshift_storage_glusterfs_storageclass_default=true	- To enable dynamic storage provisioning while cluster is getting deployed and any subsequent application pod requiring persistent volume/persistent volume claim

Key Variables	Description
openshift_storage_glusterfs_block_deploy=false	- To create a storage class. In this solution we rely on a single default storage class, as we have a single 3 node storage cluster have identical set of internal drives - Making storage class default, so that PV/PVC can be provisioned through CNS by provisioner plugin. In our solution its kubernetes.io/glusterfs

Below is the sample inventory file used in this reference architecture:

```
 [root@bastion ansible]# cat /etc/ansible/hosts
[OSEv3:children]
ansible
masters
infras
apps
etcd
nodes
lb
glusterfs
glusterfs_registry

[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-enterprise
debug_level=2
openshift_release="3.9"
openshift_enable_service_catalog=false
#ansible_become=true

# See https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
#openshift_examples_modify_imagestreams=true
openshift_disable_check=docker_image_availability

console_port=8443
openshift_debug_level="{{ debug_level }}"

openshift_node_debug_level="{{ node_debug_level | default(debug_level,true) }}"
openshift_master_debug_level="{{ master_debug_level |default(debug_level, true) }}"


openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true',
'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]

openshift_hosted_router_replicas=3
openshift_master_cluster_method=native
openshift_enable_service_catalog=false
osm_cluster_network_cidr=172.16.0.0/16
openshift_node_local_quota_per_fsgroup=512Mi

openshift_cloudprovider_vsphere_username="administrator@vsphere.local"
openshift_cloudprovider_vsphere_password="<password>"
openshift_cloudprovider_vsphere_host="ocp-vcenter.aflexpod.cisco.com"
openshift_cloudprovider_vsphere_datacenter=OCP-Datacenter
openshift_cloudprovider_vsphere_cluster=OCP-Cluster
openshift_cloudprovider_vsphere_resource_pool=ocp39
openshift_cloudprovider_vsphere_datastore="datastore4"
openshift_cloudprovider_vsphere_folder="ocp39"
openshift_cloudprovider_vsphere_template="Rhel75"
openshift_cloudprovider_vsphere_vm_network="VM Network"
openshift_cloudprovider_vsphere_vm_netmask="255.255.255.0"
openshift_cloudprovider_vsphere_vm_gateway="10.1.166.1"
openshift_cloudprovider_vsphere_vm_dns="10.1.166.9"
```

```
default_subdomain=ocp3.cisco.com

load_balancer_hostname=openshift.ocp3.cisco.com
openshift_master_cluster_hostname="{{ load_balancer_hostname }}"
openshift_master_cluster_public_hostname="{{ load_balancer_hostname }}"
openshift_master_default_subdomain="apps.ocp3.cisco.com"

os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
osm_use_cockpit=false

openshift_clock_enabled=true

# CNS registry storage
openshift_hosted_registry_replicas=3
openshift_registry_selector="region=infra"
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=30Gi

# CNS storage cluster for applications
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_block_deploy=false

# CNS storage for OpenShift infrastructure
openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_storageclass=false
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=true
openshift_storage_glusterfs_registry_block_host_vol_create=true
openshift_storage_glusterfs_registry_block_host_vol_size=100

# red hat subscription name and password
rhsub_user=<username>
rhsub_pass=<password>
rhsub_pool=<pool-id>

#registry
openshift_public_hostname=openshift.ocp3.cisco.com

[ansible]
localhost

[masters]
master-0.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.110
master-1.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.111
master-2.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.112

[infras]
infra-0.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.113
infra-1.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.114
infra-2.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.115


[apps]
app-0.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" ipv4addr=10.1.166.116
app-1.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" ipv4addr=10.1.166.117
app-2.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" ipv4addr=10.1.166.118

[etcd]
master-0.ocp3.cisco.com
master-1.ocp3.cisco.com
master-2.ocp3.cisco.com

[lb]
haproxy-0.ocp3.cisco.com openshift_node_labels="{'region': 'haproxy'}" ipv4addr=10.1.166.119

[storage]
storage-01.ocp3.cisco.com  openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.120
storage-02.ocp3.cisco.com  openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.121
storage-03.ocp3.cisco.com  openshift_node_labels="{'region': 'infra'}" ipv4addr=10.1.166.122
```

```
[nodes]
master-0.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-0.ocp3.cisco.com
master-1.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-1.ocp3.cisco.com
master-2.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-2.ocp3.cisco.com
infra-0.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
0.ocp3.cisco.com
infra-1.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
1.ocp3.cisco.com
infra-2.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
2.ocp3.cisco.com
app-0.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-0.ocp3.cisco.com
app-1.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-1.ocp3.cisco.com
app-2.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-2.ocp3.cisco.com
storage-01.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
01.ocp3.cisco.com
storage-02.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
02.ocp3.cisco.com
storage-03.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
03.ocp3.cisco.com


[glusterfs]
storage-01.ocp3.cisco.com  glusterfs_devices='[
"/dev/sde","/dev/sdf","/dev/sdg","/dev/sdh","/dev/sdi","/dev/sdj" ]'
storage-02.ocp3.cisco.com  glusterfs_devices='[
"/dev/sde","/dev/sdf","/dev/sdg","/dev/sdh","/dev/sdi","/dev/sdj" ]'
storage-03.ocp3.cisco.com  glusterfs_devices='[
"/dev/sde","/dev/sdf","/dev/sdg","/dev/sdh","/dev/sdi","/dev/sdj" ]'

[glusterfs_registry]
infra-0.ocp3.cisco.com  glusterfs_devices='[ "/dev/sdd" ]'
infra-1.ocp3.cisco.com  glusterfs_devices='[ "/dev/sdd" ]'
infra-2.ocp3.cisco.com  glusterfs_devices='[ "/dev/sdd" ]'

[root@dephost ansible]#
```

For simplicity, we used htpasswd utility for authenticating users to log into Red Hat OpenShift Container Platform. However, existing authentication methods like LDAP, AD, etc., can also be used.

For more information about various OpenShift environment variables, see: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/installation_and_configuration/#configuring-ansible

Red Hat OpenShift Container Platform Instance Creation

In this section, the following instance will be created:

- 1 x bastion, 3 x masters, 3 x infra, 3 x app, and 1 x haproxy

`etcd` requires that an odd number of cluster members exist. For collocating etcd with master nodes, three masters were chosen to support high availability and `etcd` clustering. Three infrastructure instances allow for minimal to zero downtime for applications running in the OpenShift environment. Applications instance can be one to many instances depending on the requirements of the organization.

When you scale, more infra and app nodes can be added after the initial installation.

```
# yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

```
# yum install -y python2-pyvmomi
# cd /etc/ansible
# git clone -b vmw-3.9 https://github.com/openshift/openshift-ansible-contrib
#
```

Run the following command:

Before running the prod.yaml as mentioned below, it is important that all the FQDNs are added in DNS server. Once the instance is provisioned, IP is assigned, and hostname is configured, prod.yaml playbook ssh to these nodes to perform Ansible tasks. By now having the entry in DNS server will fail those tasks by issuing error fatal: [master-0.ocp3.cisco.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: ssh: connect to host master-0.ocp3.cisco.com port 22: Connection timed out\r\n", "unreachable": true}

```
# ansible-playbook openshift-ansible-contrib/reference-architecture/vmware-ansible/playbooks/prod.yaml
```

For haproxy instance, run the following command:

```
$ ansible-playbook openshift-ansible-contrib/reference-architecture/vmware-ansible/playbooks/haproxy.yaml
```

Prod.yaml runs tasks in /etc/ansible/openshift-ansible-contrib/reference-architecture/vmware-ansible/playbooks/roles/create-vm-prod-ose/tasks/main.yaml file. Some of the values in this file are coming from inventory file /etc/ansible/hosts while some are fixed such as disk size, instance memory, and CPU. However, based on the need of different VM configuration, you may need to modify the main.yaml file.

Below is the summary of the Red Hat OpenShift Platform vSphere instance creation playbook.

The following steps do not need to be performed. Prod.yaml play book performs these tasks. However, if one or more of the tasks fail for some reason, these steps are useful for troubleshooting purposes.

1. Register with Red Hat subscription manager.
2. Enable the required repositories and disable the repositories that should not be enabled.
3. Setup vmware guest instance by installing prerequisites packages, such as open-vm-tools, PyYAML, perl, net-tools, chrony, python-six, iptables, iptables-services, and docker.

   Except docker, all other pre-requisite packages are already installed in VM template. Therefore, they will be skipped after making sure all prerequisites packages are installed.

4. Create a docker storage setup file as follows and start docker service:

```
# cat /etc/sysconfig/docker-storage-setup
DEVS="/dev/sdb"
VG="docker-vol"
DATA_SIZE="95%VG"
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME="dockerlv"
CONTAINER_ROOT_LV_MOUNT_PATH="/var/lib/docker"
```

   The /etc/sysconfig/docker-storage-setup file must be created before starting the docker service, otherwise the storage is configured using a loopback device. The container storage setup is performed on all hosts running containers, therefore masters, infrastructure, and application nodes.

5. A VMDK volume should be created for the directory of /var/lib/origin/openshift.local.volumes that is used with the perFSGroup setting at installation and with the mount option of gquota. These settings and volumes set a quota to make sure that the containers cannot grow to an unreasonable size. Create openshift-volume-quota by creating filesystem for /var/lib/origin/openshift.local.volumes

The value of OpenShift local volume size should be at least 30 GB.

6. Create fstab entry as shown below and mount the fstab entry:

```
# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Sat Feb 20 16:27:06 2010
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/rhel-root   /                       xfs     defaults        0 0
UUID=6a562c8a-f543-4851-bca1-1be97b64f18c /boot                   xfs     defaults        0 0
/dev/mapper/rhel-swap   swap                    swap    defaults        0 0
/dev/sdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0
```

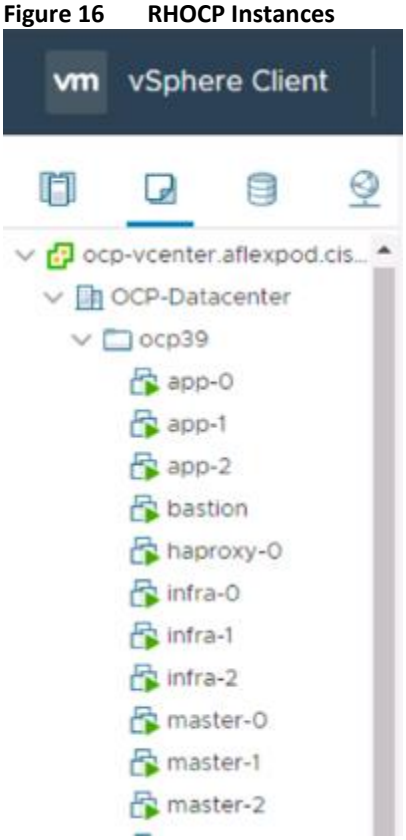
To create the master prerequisites, complete the following steps. (These are only performed on master nodes.)

1. Install git.
2. Setup etcd storage. A VMDK volume should be created on the master instances for the storage of /var/lib/etcd. Storing etcd allows the similar benefit of protecting /var but more importantly provides the ability to perform snapshots of the volume when performing etcd maintenance. Create lvm volume and create local partition on lvm lv.
3. Create the following fstab entry and mount the fstab:

```
/dev/etcd_vg/etcd_lv /var/lib/etcd xfs defaults 0 0
```

Figure 16 shows the VM instances created as a result of ansible playbook.

Figure 16 RHOCP Instances

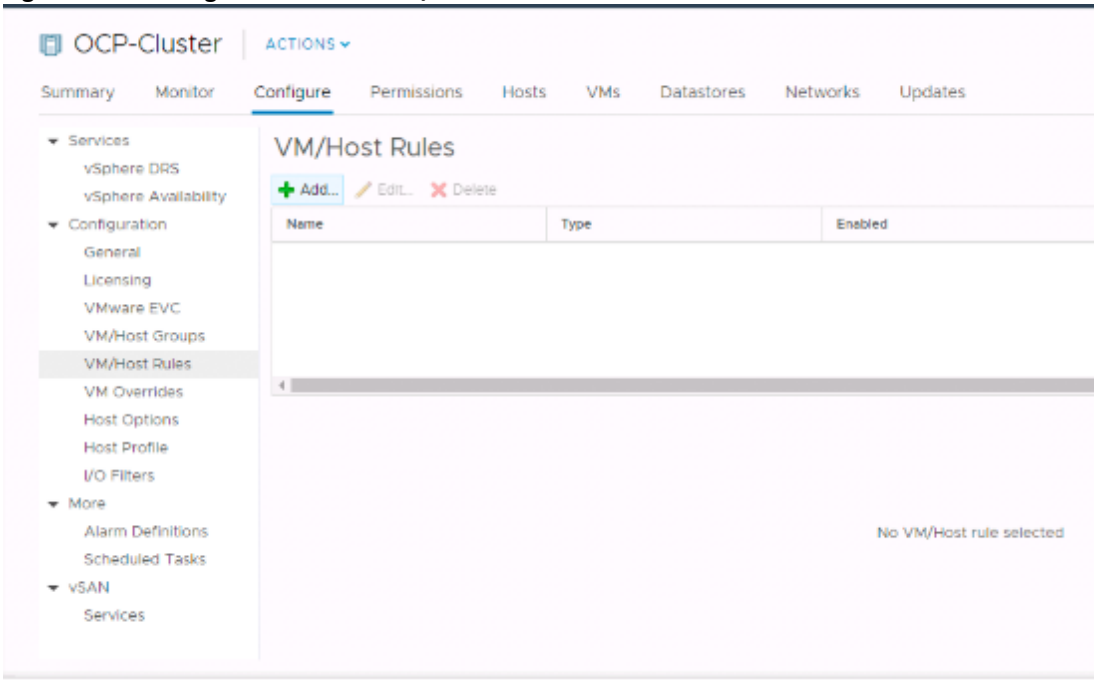


Setup DRS Anti-Affinity Rules

Create a DRS anti-affinity rules to ensure maximum availability for the cluster before you begin the deployment. To setup DRS anti-affinity rules, complete the following steps:

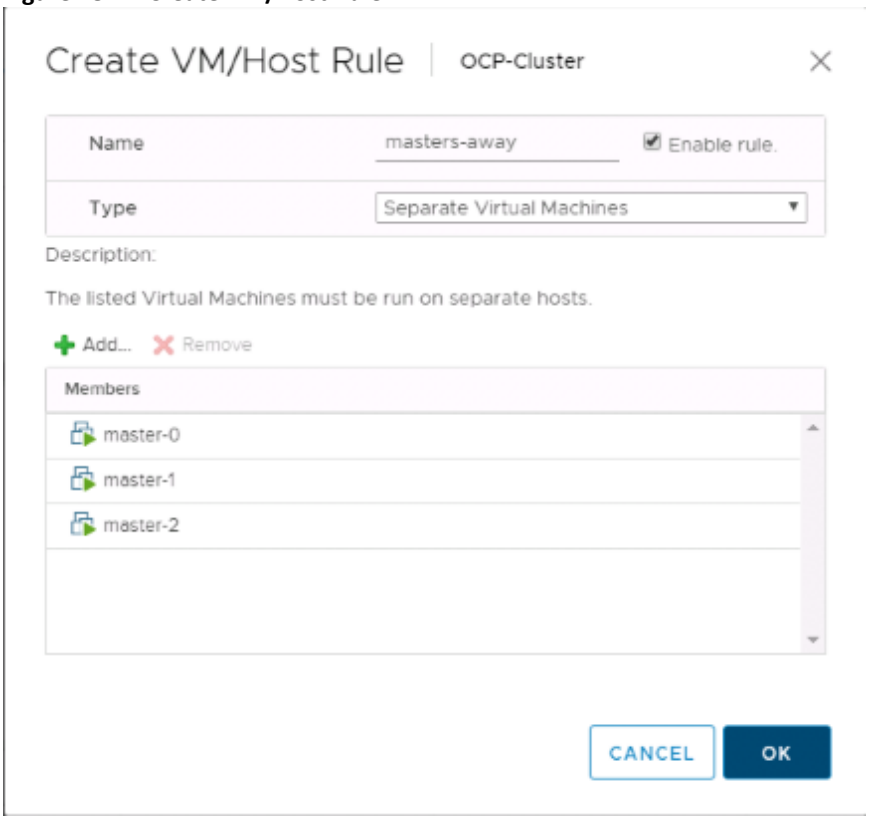
1. Open the VMware vCenter web client, select the cluster. Click Configure tab and select VM/Host Rules as shown.

Figure 17 Configure Cluster for VM/Host Rules



2. Click + Add to Create VM/Host Rule
3. Type Rule name such as master-away in this case. Make sure rule is enabled. Click + Add and select master-0, master-1, master-2 from Add Virtual Machine pop-up. Click OK to create the rule.

Figure 18 Create VM/Host Rule



See https://docs.vmware.com/en/VMware-vSphere/6.5/com.vmware.vsphere.resmgmt.doc/GUID-7297C302-378F-4AF2-9BD6-6EDB1E0A850A.html for detailed information about creating and configuring affinity rules.

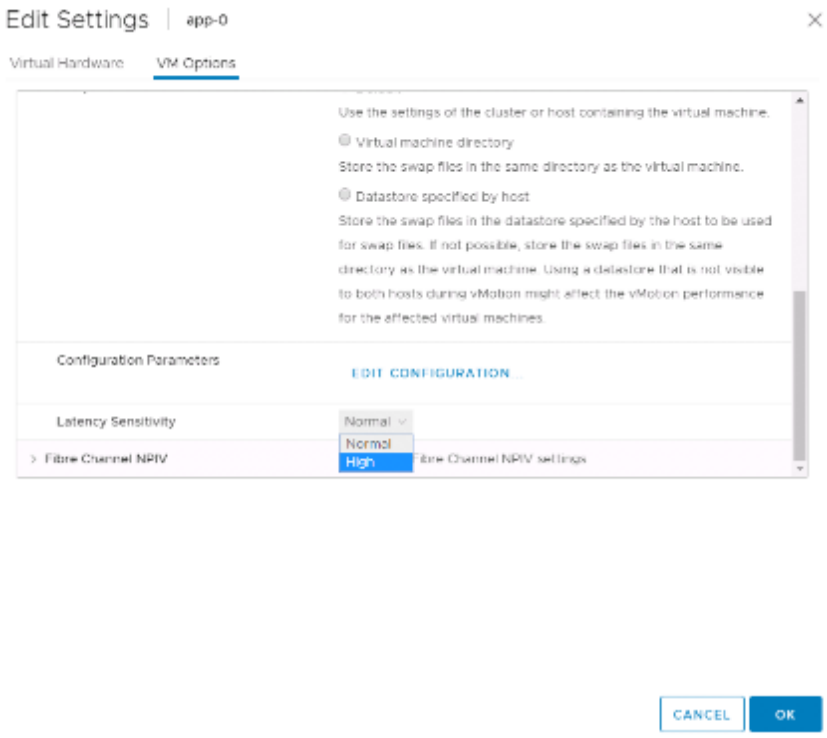
Configure VM Latency Sensitivity

Configure all of the VMs created to High VM Latency ass recommended by VMware for latency sensitive workloads.

To configure VM Latency Sensitivity, complete the following steps:

1. Open the VMware vCenter web client. Right-click VM and select Edit Settings.
2. Select VM Options tab in Edit Settings window.
3. Under VM Options tab, expand Advanced.
4. Select the 'Latency Sensitivity' dropdown and select 'High' as shown in below Figure. Click OK.

Figure 19 VMware High Latency



Red Hat OpenShift Platform Storage Node Setup

In this section, three storage nodes will be setup in Cisco UCS C240M5 Servers.

Creating Storage Profile

Storage Profiles provide a systematic way to automate the steps for provisioning Disk Groups, RAID Levels, LUNs, boot drives, hot spares, and other related resources. They are used in combination with Service Profile Templates to map the associations between logically defined storage resources and servers.

Having a Storage Profile created will reduce the task of configuring two virtual disks in the RAID Controller Option ROM or create a custom file system layout at the time of OS installation.

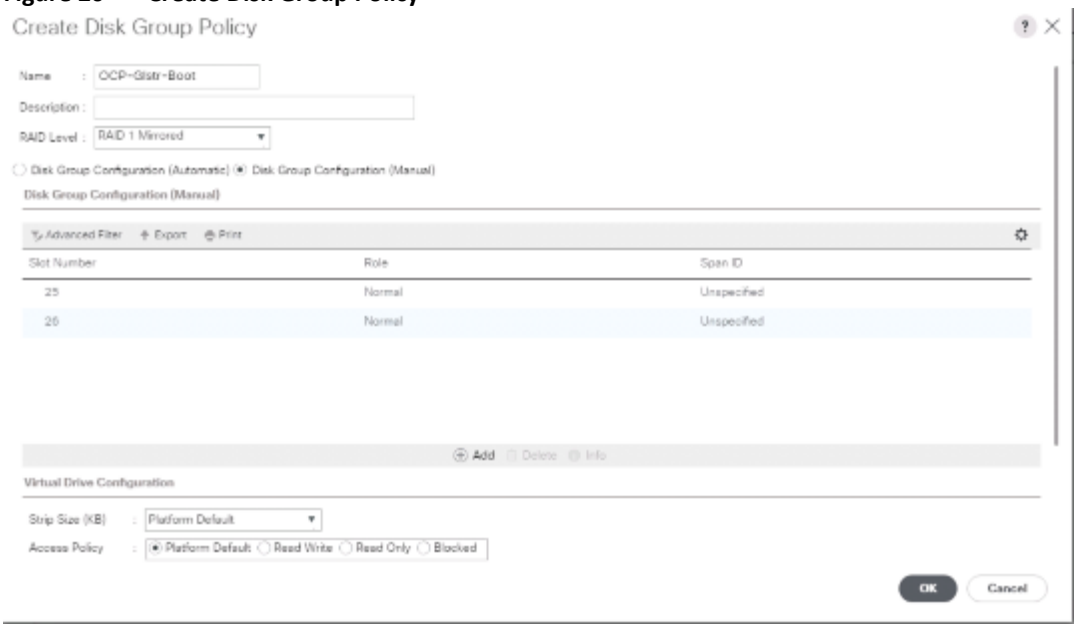
In this reference architecture, we used C240M5 servers for storage nodes. Cisco UCS C240 M5 servers support 26 drive bays. We populated 20 Intel S4500 Series SSDs to present them as JBOD disks in the solution.

We created separate storage profiles C-Series nodes.

A Storage Profile is created with two local LUNs one each for boot and data. To create a storage profile, complete the following steps:

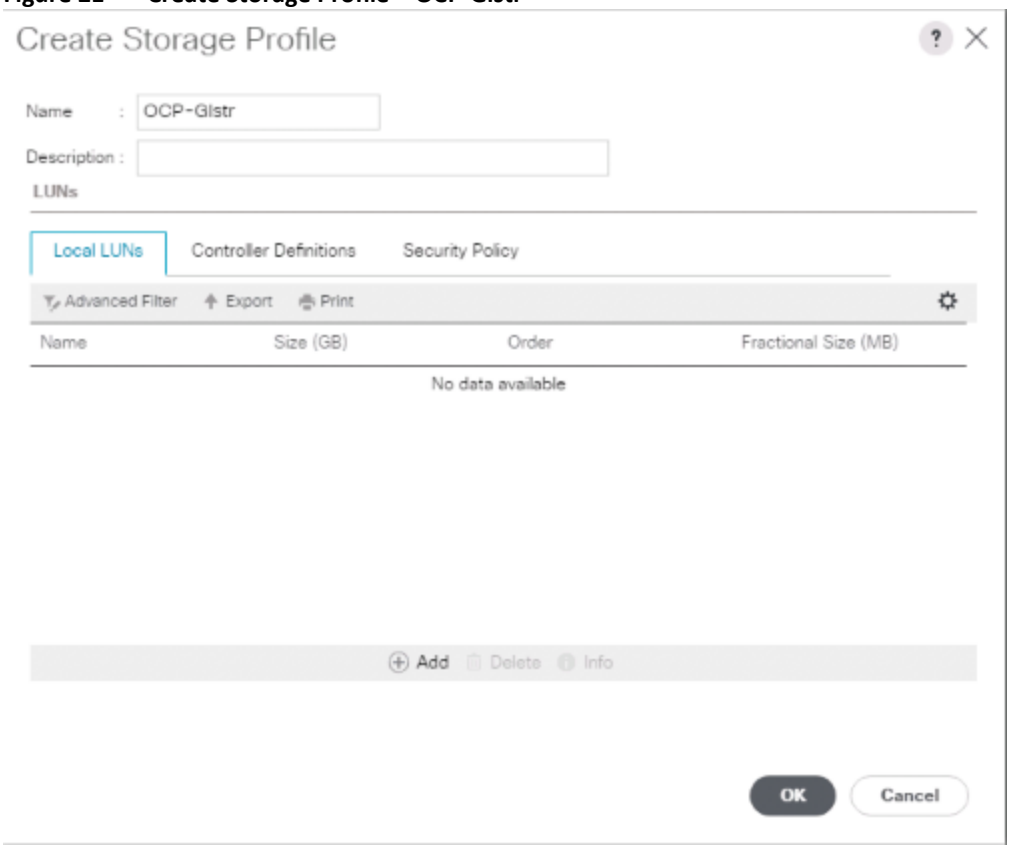
1. Click Storage in the left Navigation pane.
2. Create a disk group policy by right clicking Storage Policies > root > Disk Group Policy.
3. Enter the Disk Group name (for example, OCP-Glstr-Boot). Keep the RAID Level as RAID 1 Mirrored.
4. Select Disk Group Configuration (Manual) radio button. Click + to add two slots; slot 25 and 26.
5. Click OK.

Figure 20 Create Disk Group Policy



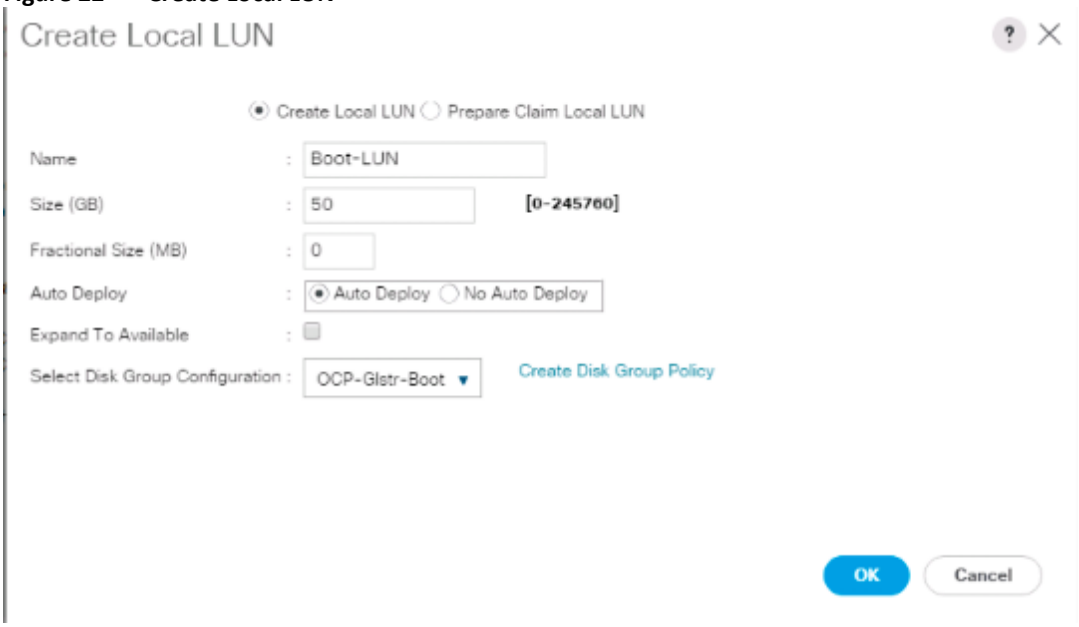
6. Select Storage Profiles.
7. Right-click Storage Profiles and select Create Storage Profile.
8. Enter the name for the Storage Profile (for example, OCP-Glstr for Rack servers C240M5) as shown.

Figure 21 Create Storage Profile – OCP-Glstr



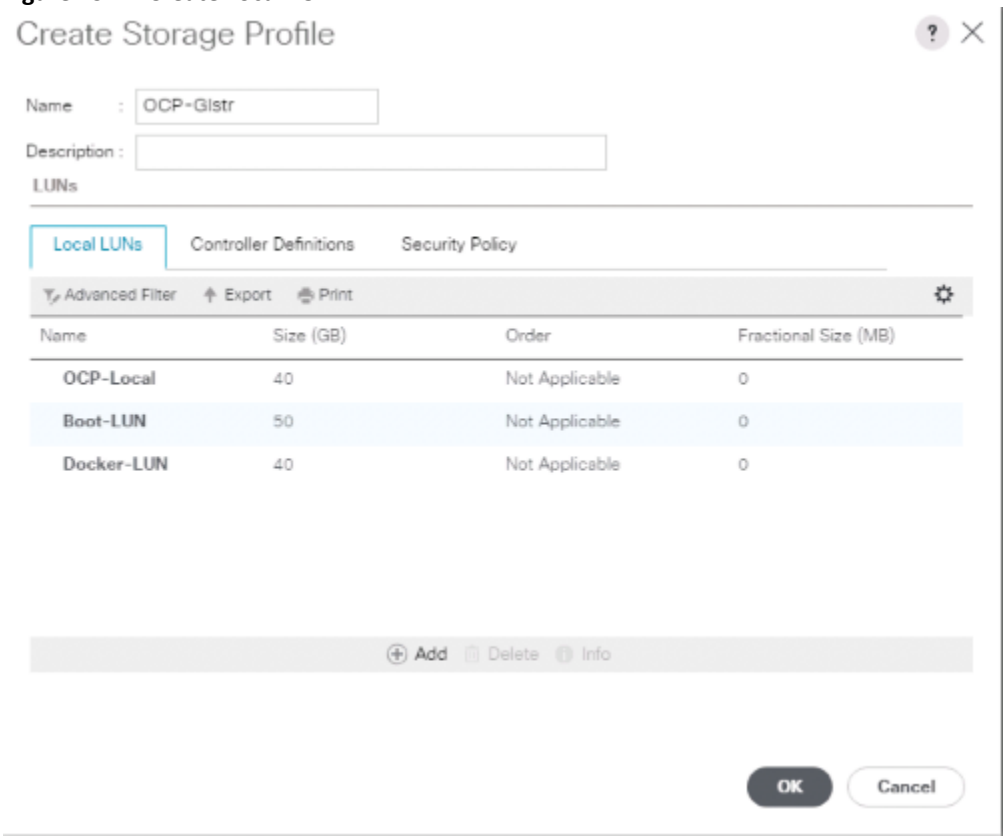
9. Click Add to in the Local LUN tab. Select Create Local LUN radio button. Specify the LUN name for example Boot-LUN, size 50GB, and select OCP-Glstr-Boot from Select Disk Group Configuration drop down as shown.
10. Click OK to create Boot-LUN.

Figure 22 Create Local LUN



11. Repeat step 9 to create Docker LUN and OCP-LUN. Final output will look like as shown below.
12. Click OK to create storage profile with three LUNs; Boot, Docker, and OpenShift local.

Figure 23 Create Local LUN

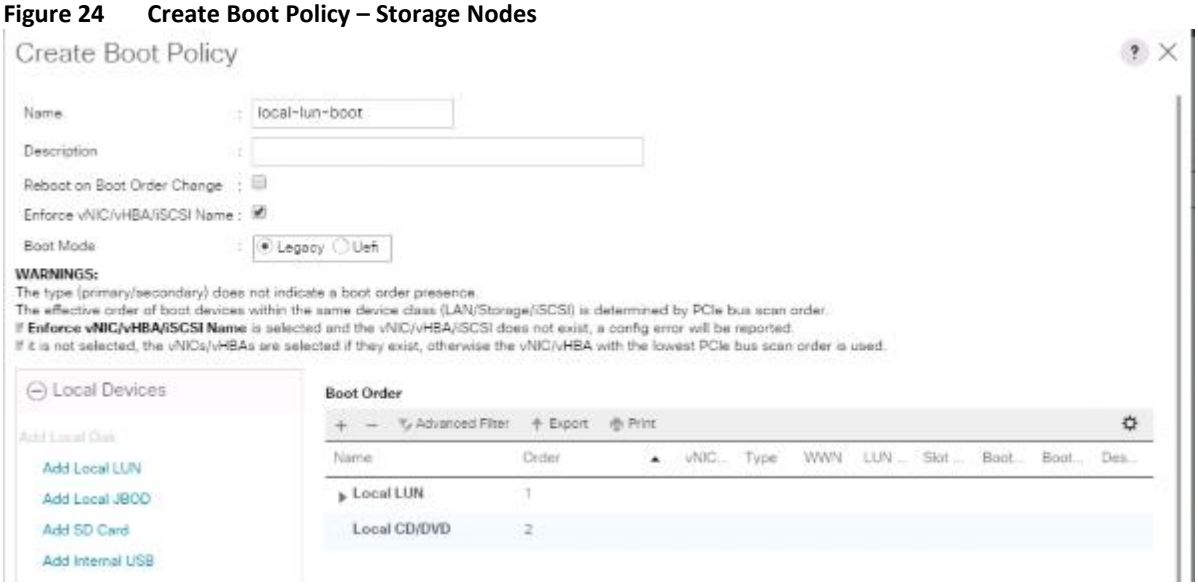


Boot Policy for Storage Node

To create a boot policy for storage node, complete the following steps:

1. Click Servers in the left navigation pane. Expand Policies > root > Boot Policies. Right-click Boot Policies.
2. Provide a boot policy name (for example. Local-lun-boot).
3. Add Local LUN and Local CD/DVD in the boot order as shown below.
4. Click OK to create the boot policy.

Figure 24 Create Boot Policy – Storage Nodes

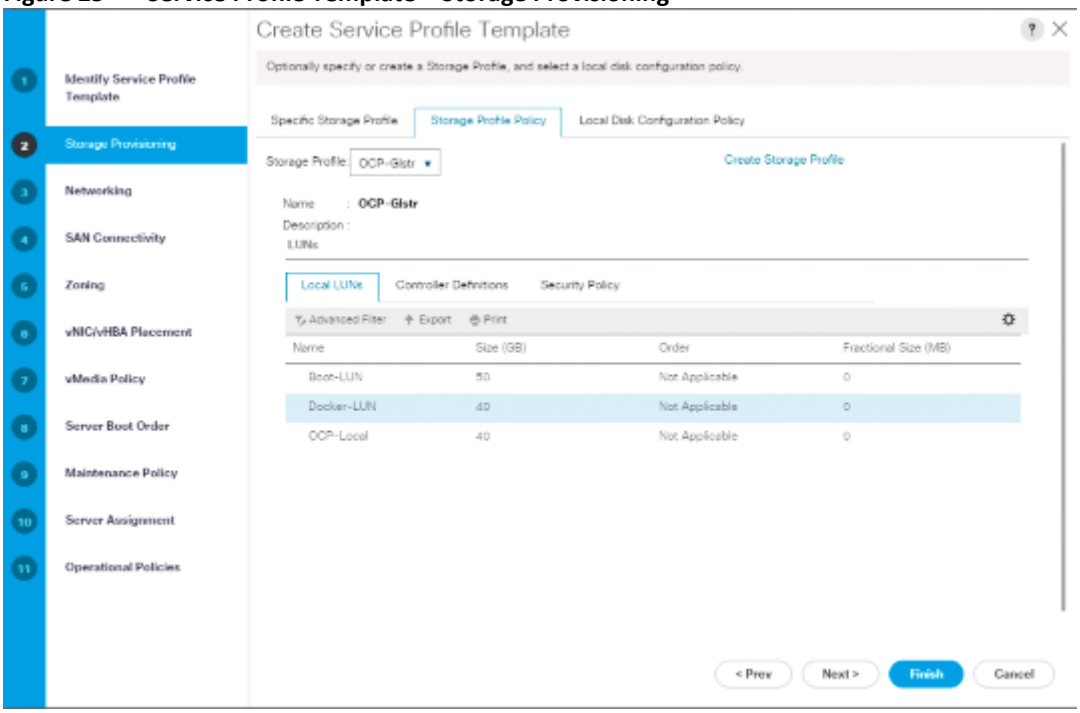


Boot, docker, and openshift volume LUNs can also be provided from already existing SAN. Boot Policy should be configured by adding iSCSI Boot if IP based storage is configured using iSCSI or by adding SAN boot if vHBAs are configured in the existing environment.

Service Profile Template for Storage Nodes

If Storage profile is used for booting the storage nodes, storage nodes service profile template, then Storage provisioning should be configured with storage profile as shown in Figure 25.

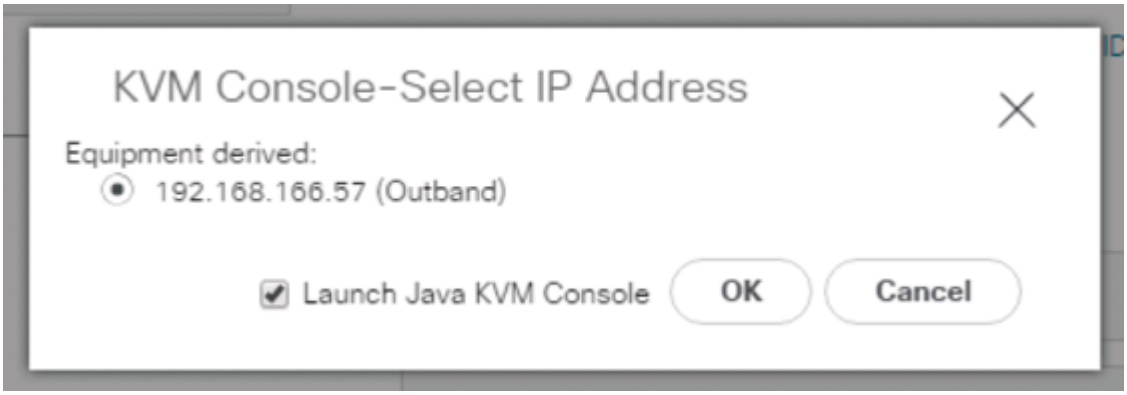
Figure 25 Service Profile Template – Storage Provisioning



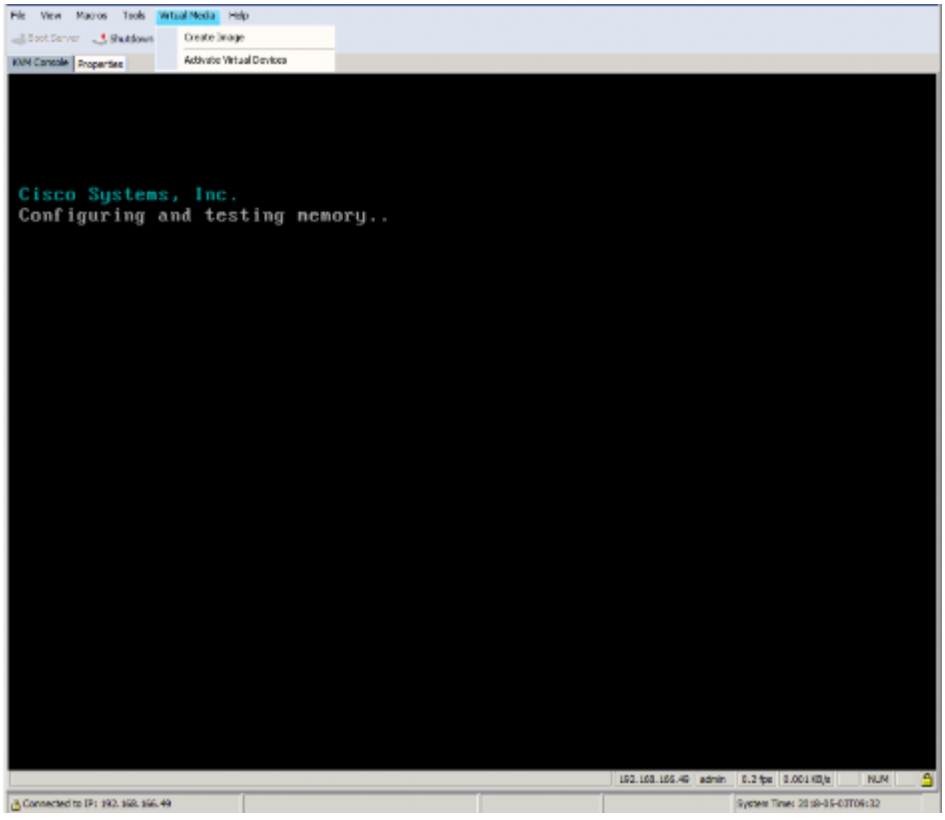
Installation of Red Hat Enterprise Linux Operating System in Storage Nodes

To install the Operating System on the storage nodes, complete the following steps:

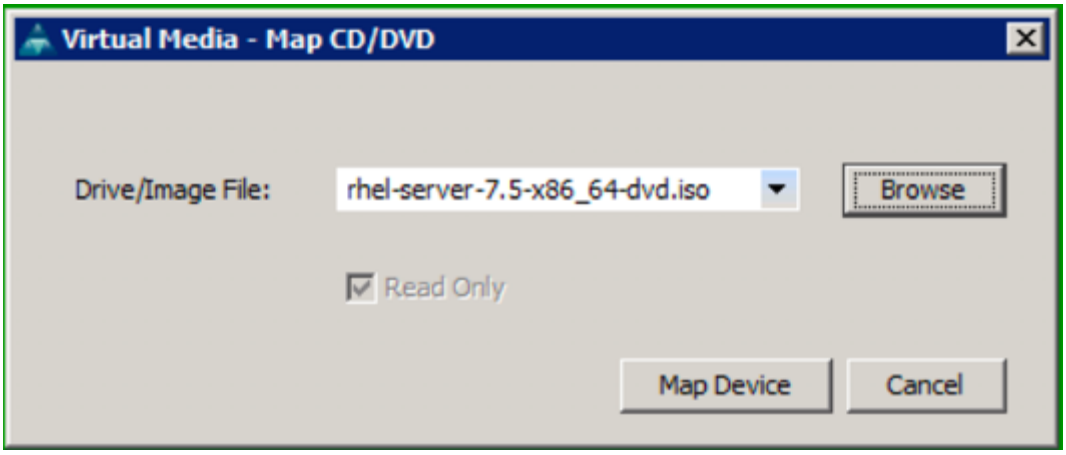
1. Download Red Hat Enterprise Linux 7.5 from http://access.redhat.com.
2. Complete the following to install Red Hat Enterprise Linux 7.5 OS.
3. From Cisco UCS Manager, click Servers > Service Profiles > Storage-01
4. Launch the KVM console from General tab. Click >> to launch Java JVM Console. Click OK.



5. In the KVM Console, select Activate Virtual Devices by clicking the Virtual Media drop-down list.

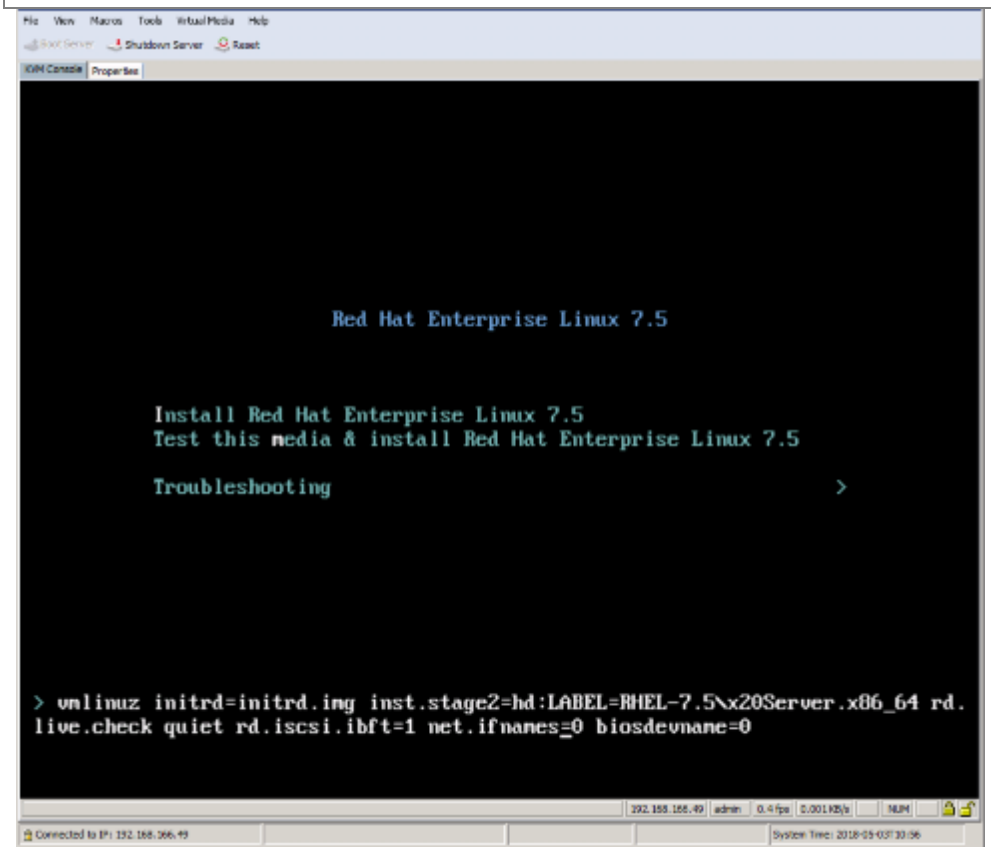


6. When the virtual devices get activated, select Map CD/DVD from the Virtual Media drop-down list.
7. Click Browse in the Virtual Media – Map CD/DVD pop-up window.
8. Locate the Red Hat Enterprise Linux Server 7.5 installer ISO image file and click Map Device as shown.

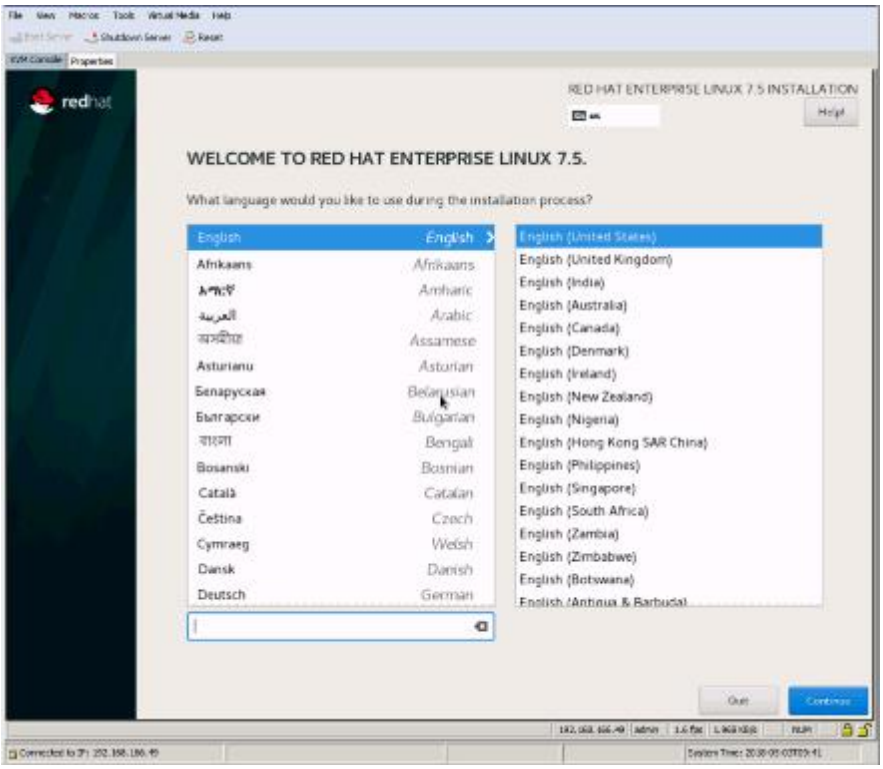


9. The image gets mapped to CD/DVD.
10. Click the Reset on the menu bar. Click Power Cycle. Click OK.
11. In KVM window, monitor the reboot. On reboot, the machine detects the presence of the Red Hat Enterprise Linux Server 7.5 install media. At the prompt press F6 to select the Boot Device. Select Cisco vKVM-Mapped vDVD and press Enter key.
12. Server will detect the installation media of RHEL 7.5
13. Press tab key for full configuration
14. Type the following and press the Enter key to proceed to installation.

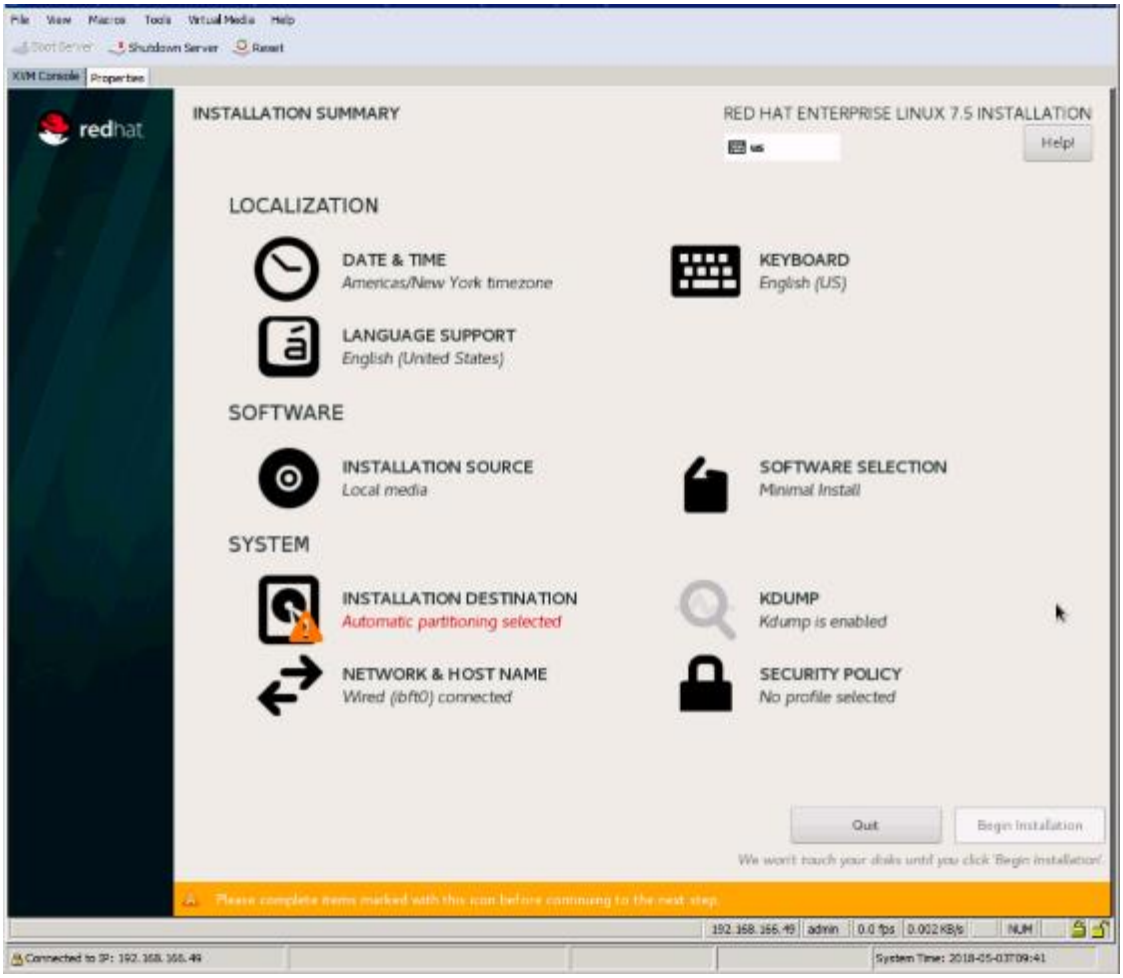
```
rd.iscsi.ibft=1 net.ifnames=0 biosdevname=0
```



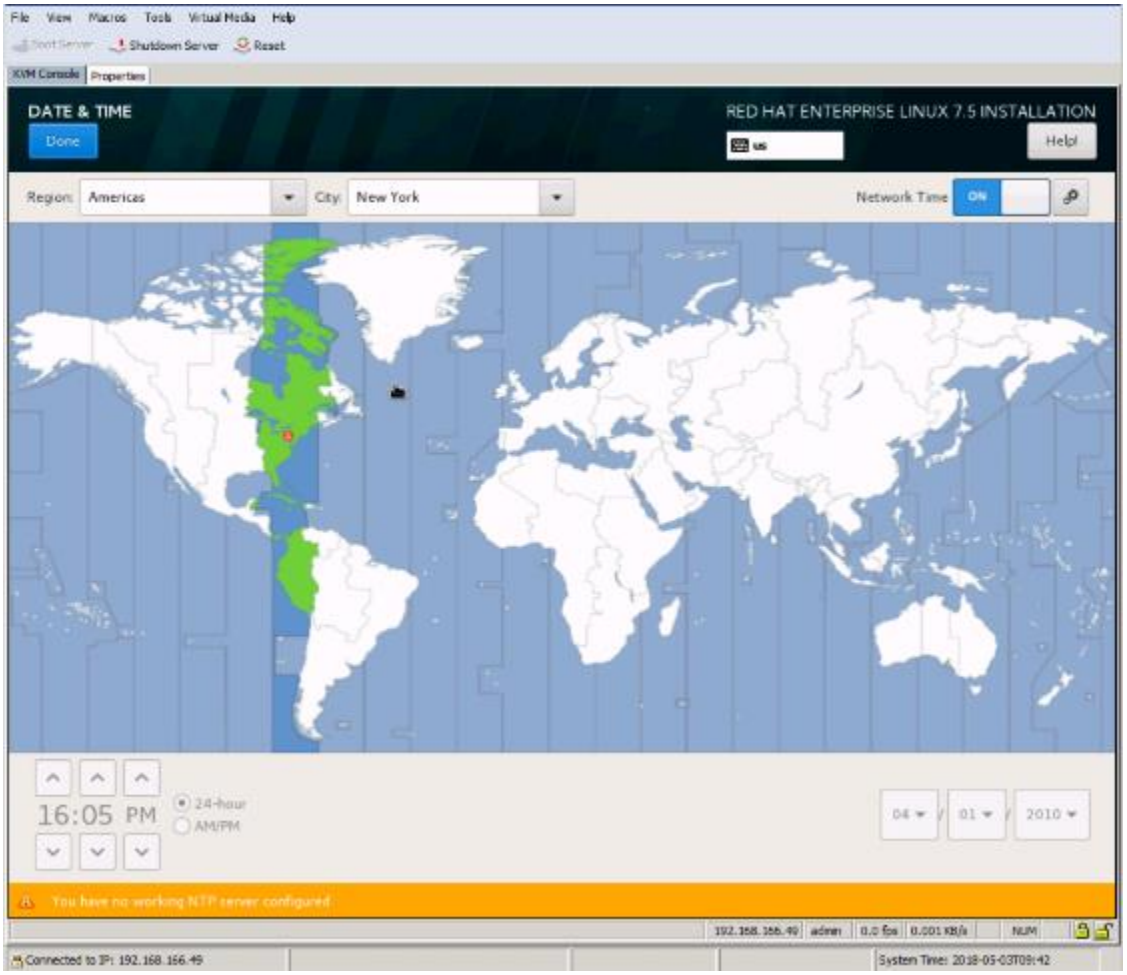
15. Select Language and click Continue.



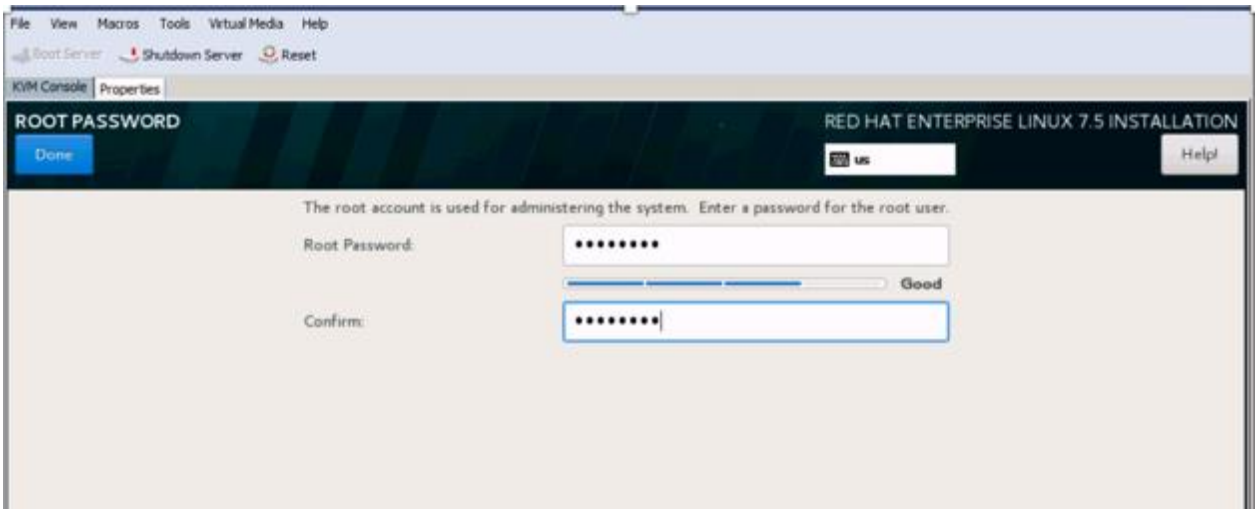
16. Click DATE & TIME.



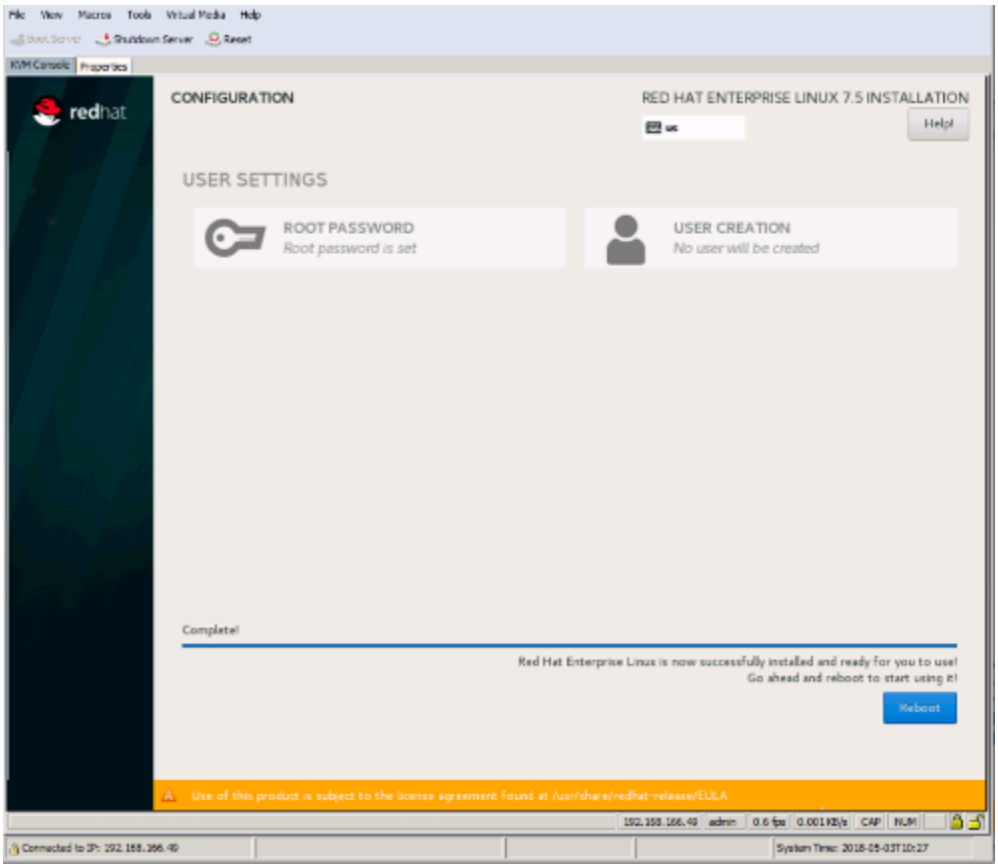
17. Select Region and City in DATE & TIME screen and click Done.



18. Click INSTALLATION DESTINATION. Select boot LUN. Click Done.
19. Click Done on device selection for installation destination.
20. Click Done and click Begin Installation.
21. While the installation in is progress, click ROOT PASSWORD to assert a password and then click User Creation to set user credentials.



22. When the installation is complete, click Reboot.



Configure Storage Node Interfaces for RHOCP

To configure the storage node interfaces for RHOCP, complete the following steps:

1. Add vlan tag interface in all three storage nodes as below. Assign the respective IP addresses to each storage node:

```
# nmcli con add type vlan ifname vlan117 dev eth0 id 117 ip4 10.1.166.120/24 \
gw4 10.1.166.1
# nmcli con show
[root@storage-01 ~]# nmcli con show
NAME           UUID                                   TYPE      DEVICE
eth0           5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03   ethernet  eth0
vlan-vlan117   0cfc3855-bf9d-4c10-952b-fc284d4cd885   vlan      vlan117
```

2. Interface configuration will look like as below:

```
[root@storage-01 ~]#
[root@storage-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
[root@storage-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-vlan-vlan117
VLAN=yes
TYPE=Vlan
PHYSDEV=eth0
VLAN_ID=117
REORDER_HDR=yes
```

```
GVRP=no
MVRP=no
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.1.166.120
PREFIX=24
GATEWAY=10.1.166.1
DNS1=10.1.166.120
DNS2=10.1.166.9
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=vlan-vlan117
UUID=0cfc3855-bf9d-4c10-952b-fc284d4cd885
DEVICE=vlan117
ONBOOT=yes
NM_CONTROLLED=yes
[root@storage-01 ~]#
```

3. Configure host name for storage nodes by running the following command:

```
# hostnamectl set-hostname storage-01.ocp3.cisco.com
```

Creating an SSH Keypair for Ansible

After all the VMs and Cisco UCS C240 Servers are deployed successfully. The VMware infrastructure requires an SSH key on the VMs for Ansible's use.

In the bastion instance, complete the following steps:

1. Generate a new SSH key to be used for authentication.

```
$ ssh-keygen -N '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/root/.ssh'.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:aaQHUf2rKHWvwwl4RmYcmCHswoouu3rdZiSH/BYgzBg root@ansible-test
The key's randomart image is:
+---[RSA 2048]----+
|   .. o=..       |
|E   ..o.. .      |
| * .  .... .     |
|. * o  +=.  .    |
|.. + o.=S    .   |
|o   + =o= . .    |
|.  . * = = +     |
|... . B . =  .   |
+----[SHA256]-----+
```

2. Add the ssh keys to all the deployed virtual machines and storage bare-metal nodes via ssh-copy-id or to the template prior to deployment as shown below:

```
[root@bastion ansible]# MASTERS="master-0 master-1 master-2"
[root@bastion ansible]# INFRA_NODES="infra-0 infra-1 infra-2"
[root@bastion ansible]# APP_NODES="app-0 app-1 app-2"
[root@bastion ansible]# STORAGE_NODES="storage-01 storage-02 storage-03"
[root@bastion ansible]# ALL_HOSTS="$MASTERS $INFRA_NODES $APP_NODES $STORAGE_NODES"

[root@bastion ansible]# for H in $ALL_HOSTS ; do   ssh-copy-id -i ~/.ssh/id_rsa.pub root@$H; done
```

```
```

3. Verify ssh by the running the following command for Ansible. Ensure all virtual nodes (master, infra, and app) and physical storage nodes are accessible via Ansible through fully qualified domain names.

```
# ssh master-0
```

4. Make sure there is connectivity to all instances via bastion node as follows:

```
[root@bastion ansible]# ansible all -m ping
storage-02.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
storage-01.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
storage-03.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
master-0.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
master-1.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
localhost | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
infra-0.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
master-2.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
infra-1.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
app-0.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
infra-2.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
app-1.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
```

```
haproxy-0.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
app-2.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
[root@bastion ansible]#
```

Configure and Install Prerequisites for Storage Nodes

To verify the following prerequisites in storage nodes before proceeding for RHOCP deployment, complete the following steps:

1. Verify storage nodes are accessible via Ansible.

```
[root@bastion ansible]# ansible storage -m ping
storage-02.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
storage-03.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
storage-01.ocp3.cisco.com | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
```

2. Register storage nodes with Red Hat subscription and enable required repositories

```
# ansible storage -m command -a "subscription-manager register --username <user-name> --password
'<password>'"
# ansible all -m command -a "subscription-manager attach --pool=<pool-id>"
# ansible storage -m command -a "subscription-manager repos --disable='*'"
# ansible storage -m command -a "subscription-manager repos --enable="rhel-7-server-rpms" --enable="rhel-
7-server-extras-rpms" --enable="rhel-7-server-ose-3.9-rpms" --enable="rhel-7-fast-datapath-rpms" --
enable="rhel-7-server-ansible-2.4-rpms""
```

If the system is slow, registration will fail with the following:

Registering to: subscription.rhsm.redhat.com:443/subscription. Unable to verify server's identity: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:579) non-zero return code.

3. Install prerequisites packages such as PyYAML, perl, net-tools, chrony, python-six, iptables, iptables-services, docker:

```
# ansible storage -m command -a "yum install -y PyYAML perl net-tools chrony python-six iptables
iptables-services"
```

4. Create a docker-storage-setup file in bastion node with the following contents and copy it in /etc/sys-config/ in all storage nodes:

```
# vi docker-storage-setup
DEVS="/dev/sdb"
```

```
VG="docker-vol"
DATA_SIZE="95%VG"
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME="dockerlv"
CONTAINER_ROOT_LV_MOUNT_PATH="/var/lib/docker"
# for H in $STORAGE_NODES ; do echo "Copying to-->"$H; scp ~/docker-storage-setup
root@$H:/etc/sysconfig/.; done
```

5. Setup OpenShift local volume:

```
# ansible storage -m command -a  "mkfs -t xfs /dev/sdc"
```

6. Create an fstab entry for OpenShift local volume:

```
# ansible storage -m command -a "echo '/dev/sdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0'
>> /etc/fstab"
[root@bastion ansible]# ansible storage -m command -a "cat /etc/fstab"
storage-03.ocp3.cisco.com | SUCCESS | rc=0 >>

#
# /etc/fstab
# Created by anaconda on Wed Aug  8 15:54:42 2018
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/rhel-root   /                       xfs     defaults        0 0
UUID=e561477b-4e29-4cb3-8394-d20b42fcef5f /boot                   xfs     defaults        0 0
/dev/mapper/rhel-home   /home                   xfs     defaults        0 0
#/dev/mapper/rhel-swap   swap                    swap    defaults        0 0
/dev/sdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0

--- ouput truncated.
```

7. Mount the file system mentioned in fstab:

```
#ansible storage -m command -a "mount -a"
```

8. Start and verify the docker service:

```
# ansible storage -m command -a "systemctl start docker"
# ansible storage -m command -a "systemctl status docker"
```

Instance Verification

It can be useful to check for potential issues or misconfigurations in the instances before continuing the installation process. Connect to every instance using the deployment host and verify the disks are properly created and mounted:

```
$ ssh master-0.ocp3.cisco.com
$ lsblk
$ sudo journalctl
$ free -m
$ cat /etc/sysconfig/docker-storage-setup
$ cat /etc/fstab
$ sudo yum repolist
```

Where the instance is master-0.ocp3.cisco.com

For reference, the following is an example output of lsblk for the master nodes:

```
 [root@master-0 ~]# lsblk
NAME                    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                       8:0    0   60G  0 disk
├─sda1                    8:1    0    1G  0 part /boot
└─sda2                    8:2    0   49G  0 part
```

```
  ├─rhel-root             253:0    0 45.1G  0 lvm  /
  └─rhel-swap             253:1    0  3.9G  0 lvm  [SWAP]
sdb                         8:16   0   40G  0 disk
└─sdb1                      8:17   0   40G  0 part
  └─docker--vol-dockerlv 253:2    0   40G  0 lvm  /var/lib/docker
sdc                         8:32   0   40G  0 disk /var/lib/origin/openshift.local.volumes
sdd                         8:48   0   40G  0 disk
└─etcd_vg-etcd_lv       253:3    0   38G  0 lvm  /var/lib/etcd
sr0                      11:0    1  4.3G  0 rom
[root@master-0 ~]#
```

For reference, the following is an example of output of lsblk for the infra and app nodes:

```
 [root@infra-0 ~]# lsblk
NAME                    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                       8:0    0   60G  0 disk
├─sda1                    8:1    0    1G  0 part /boot
└─sda2                    8:2    0   49G  0 part
  ├─rhel-root           253:0    0 45.1G  0 lvm  /
  └─rhel-swap           253:1    0  3.9G  0 lvm  [SWAP]
sdb                       8:16   0   40G  0 disk
└─sdb1                    8:17   0   40G  0 part
  └─docker--vol-dockerlv 253:2    0   40G  0 lvm  /var/lib/docker
sdc                       8:32   0   40G  0 disk /var/lib/origin/openshift.local.volumes
sdd                       8:48   0   300G 0 disk
sr0                      11:0    1  4.3G  0 rom
[root@infra-0 ~]#
```

Red Hat OpenShift Container Platform Prerequisites Playbook

The Red Hat OpenShift Container Platform Ansible installation provides a playbook to ensure all prerequisites are met prior to the installation of Red Hat OpenShift Container Platform. This includes steps such as registering all the nodes with Red Hat Subscription Manager and setting up the docker on the docker volumes. The playbook also skips if anything is found already configured and installed. Playbook will also report any error if occurs.

Run the prerequisites ansible-playbook on the bastion node to help ensure all the prerequisites are met using prerequisites.yml playbook as shown in Figure 26:

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

Figure 26 Prerequisites Playbook

```
PLAY RECAP *********************************************************************************************************
app-0.ocp3.cisco.com       : ok=67   changed=3    unreachable=0    failed=0
app-1.ocp3.cisco.com       : ok=67   changed=3    unreachable=0    failed=0
app-2.ocp3.cisco.com       : ok=67   changed=3    unreachable=0    failed=0
haproxy-0.ocp3.cisco.com   : ok=42   changed=2    unreachable=0    failed=0
infra-0.ocp3.cisco.com     : ok=67   changed=3    unreachable=0    failed=0
infra-1.ocp3.cisco.com     : ok=67   changed=3    unreachable=0    failed=0
infra-2.ocp3.cisco.com     : ok=67   changed=3    unreachable=0    failed=0
localhost                  : ok=13   changed=0    unreachable=0    failed=0
master-0.ocp3.cisco.com    : ok=80   changed=4    unreachable=0    failed=0
master-1.ocp3.cisco.com    : ok=71   changed=3    unreachable=0    failed=0
master-2.ocp3.cisco.com    : ok=71   changed=3    unreachable=0    failed=0
storage-01.ocp3.cisco.com  : ok=72   changed=20   unreachable=0    failed=0
storage-02.ocp3.cisco.com  : ok=70   changed=20   unreachable=0    failed=0
storage-03.ocp3.cisco.com  : ok=70   changed=20   unreachable=0    failed=0


INSTALLER STATUS ***************************************************************************************************
Initialization             : Complete (0:04:15)
```

Deploying Red Hat OpenShift Container Platform

With the prerequisites met, the focus shifts to the installation of Red Hat OpenShift Container Platform. The installation and configuration is done via a series of Ansible playbooks and roles provided by the OpenShift RPM packages.

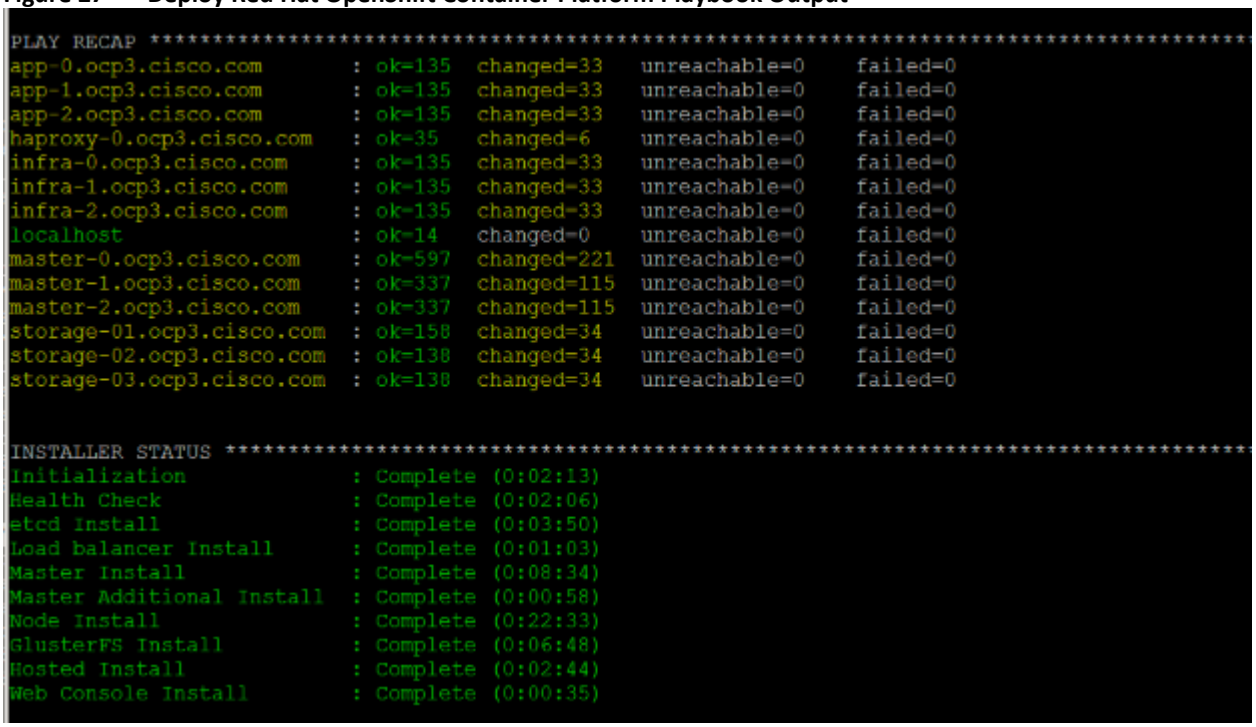
Deploy Red Hat OpenShift Container Platform by running the installer playbook as shown below:

```
# ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

The playbook will run through several tasks while installing Red Hat OpenShift Container Platform. It will report any errors if occur. Figure 27 shows the successful output of the playbook. For detailed information about the tasks, use the -vvv option as shown below:

```
# ansible-playbook -vvv /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

Figure 27 Deploy Red Hat OpenShift Container Platform Playbook Output



Functional Validation

In this section, you will go through a series of deployment verification tasks; these include both UI and CLI tasks. After the installation has completed successfully, complete the following steps:

1. Verify if the master is started and nodes are registered and reporting `Ready` status. On one of the master node, run the following as root:

```
[root@master-1 ~]# oc get nodes
NAME                      STATUS    ROLES     AGE       VERSION
app-0.ocp3.cisco.com      Ready     compute   26d       v1.9.1+a0ce1bc657
app-1.ocp3.cisco.com      Ready     compute   26d       v1.9.1+a0ce1bc657
app-2.ocp3.cisco.com      Ready     compute   26d       v1.9.1+a0ce1bc657
infra-0.ocp3.cisco.com    Ready     <none>    26d       v1.9.1+a0ce1bc657
infra-1.ocp3.cisco.com    Ready     <none>    26d       v1.9.1+a0ce1bc657
infra-2.ocp3.cisco.com    Ready     <none>    26d       v1.9.1+a0ce1bc657
master-0.ocp3.cisco.com   Ready     master    26d       v1.9.1+a0ce1bc657
```

```
master-1.ocp3.cisco.com     Ready     master    26d       v1.9.1+a0ce1bc657
master-2.ocp3.cisco.com     Ready     master    26d       v1.9.1+a0ce1bc657
storage-01.ocp3.cisco.com   Ready     compute   26d       v1.9.1+a0ce1bc657
storage-02.ocp3.cisco.com   Ready     compute   26d       v1.9.1+a0ce1bc657
storage-03.ocp3.cisco.com   Ready     compute   26d       v1.9.1+a0ce1bc657
[root@master-1 ~]#
```

In order to provide `admin` user cluster-admin role, execute following command on the master node:
[root@OCP-Mstr-1 ~]# oc adm policy add-cluster-role-to-user cluster-admin admin --as=system:admin
cluster role "cluster-admin" added: "admin"
This allows admin user to act as cluster-admin user and can manage cluster wide projects.

2. Verify services:

```
[root@master-1 ~]# oc get svc
NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                   AGE
docker-registry                ClusterIP   172.30.200.159   <none>        5000/TCP                  26d
glusterfs-registry-endpoints   ClusterIP   172.30.97.152    <none>        1/TCP                     26d
kubernetes                     ClusterIP   172.30.0.1       <none>        443/TCP,53/UDP,53/TCP     26d
registry-console               ClusterIP   172.30.122.27    <none>        9000/TCP                  26d
router                         ClusterIP   172.30.253.218   <none>        80/TCP,443/TCP,1936/TCP   26d
[root@master-1 ~]#
```

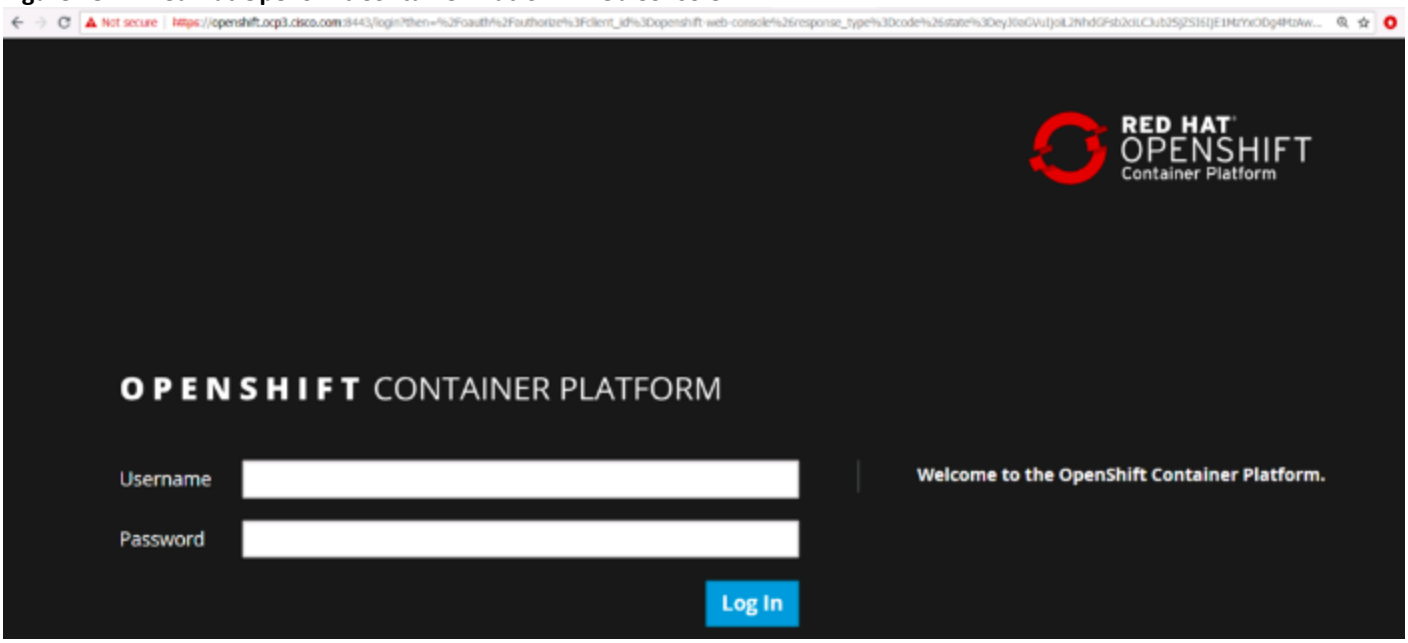
3. Verify hxproxy-0 instance is up and running and /etc/haproxy/haproxy.cfg file matches with the following and haproxy service is running before validating web console. It is also important to note that openshift.ocp3.cisco.com is resolving to haproxy IP address in DNS records.

```
 [root@haproxy-0 ~]# cat /etc/haproxy/haproxy.cfg
# Global settings
#---------------------------------------------------------------------
global
    maxconn     20000
    log         /dev/log local0 info
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    user        haproxy
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
#    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
#    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          300s
    timeout server          300s
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 20000
```

```
listen stats
    bind :9000
    mode http
    stats enable
    stats uri /

frontend  main_80
     bind *:80
     default_backend router80
     mode tcp
     option tcplog

backend router80
    balance source
    mode tcp
    # INFRA_80
    server infra-0.ocp3.cisco.com 10.1.166.113:80 check
    server infra-1.ocp3.cisco.com 10.1.166.114:80 check
    server infra-2.ocp3.cisco.com 10.1.166.115:80 check


frontend main_443
     bind *:443
     default_backend router443
     mode tcp
     option tcplog

backend router443
     balance source
     mode tcp
#    INFRA_443
     server infra-0.ocp3.cisco.com 10.1.166.113:443 check
     server infra-1.ocp3.cisco.com 10.1.166.114:443 check
     server infra-2.ocp3.cisco.com 10.1.166.115:443 check



frontend  main_8443
    bind *:8443
    default_backend mgmt8443
    mode tcp
    option tcplog

backend mgmt8443
    balance source
    mode tcp
    # MASTERS_8443
    server      master-0.ocp3.cisco.com 10.1.166.110:8443 check
    server      master-1.ocp3.cisco.com 10.1.166.111:8443 check
    server      master-2.ocp3.cisco.com 10.1.166.112:8443 check
[root@haproxy-0 ~]#
```

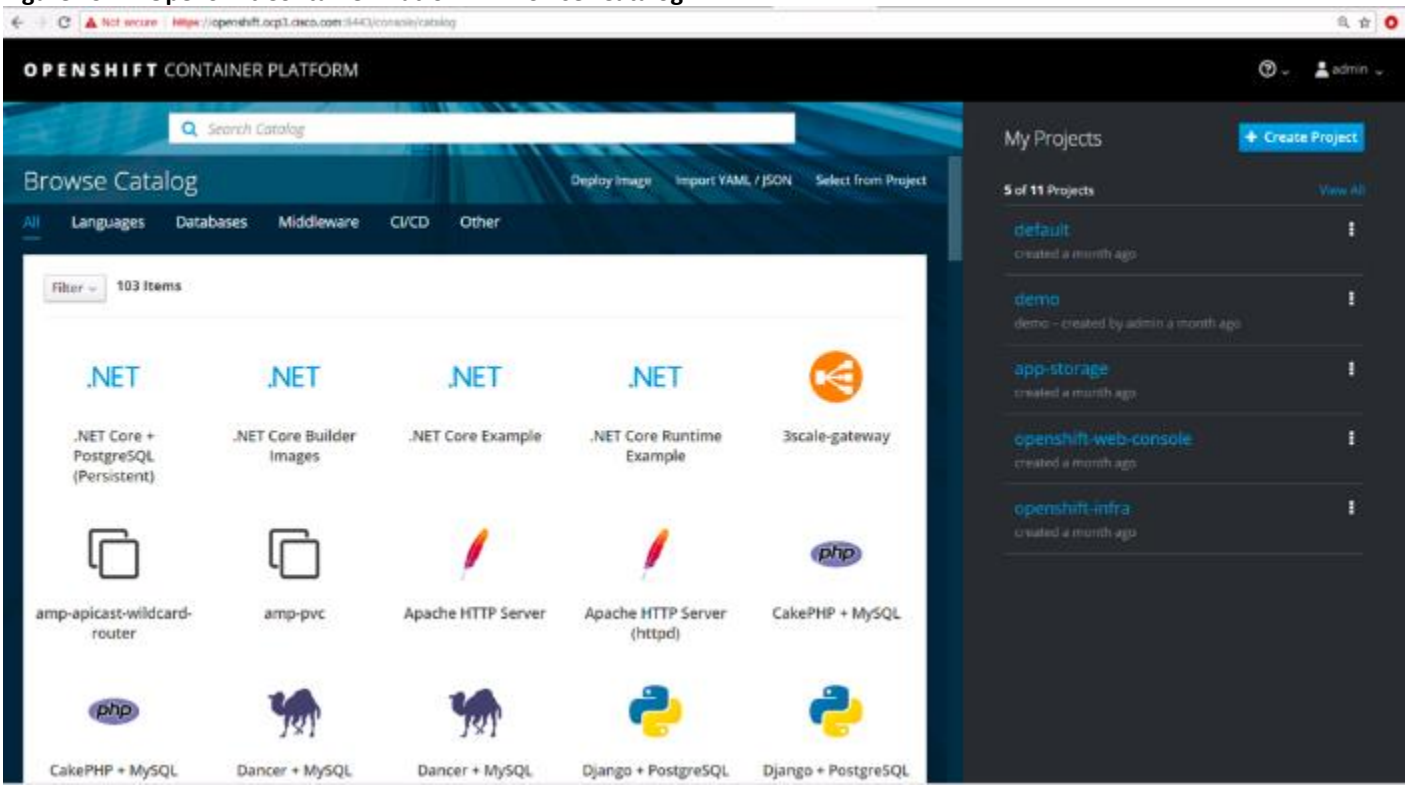
4. Verify if the web console is installed correctly. Use `openshift_master_cluster_public_hostname` value, as defined in the inventory file, with the URL for web-console access: https://openshift.ocp3.cisco.com:8443.

Figure 28 Red Hat OpenShift Container Platform Web Console



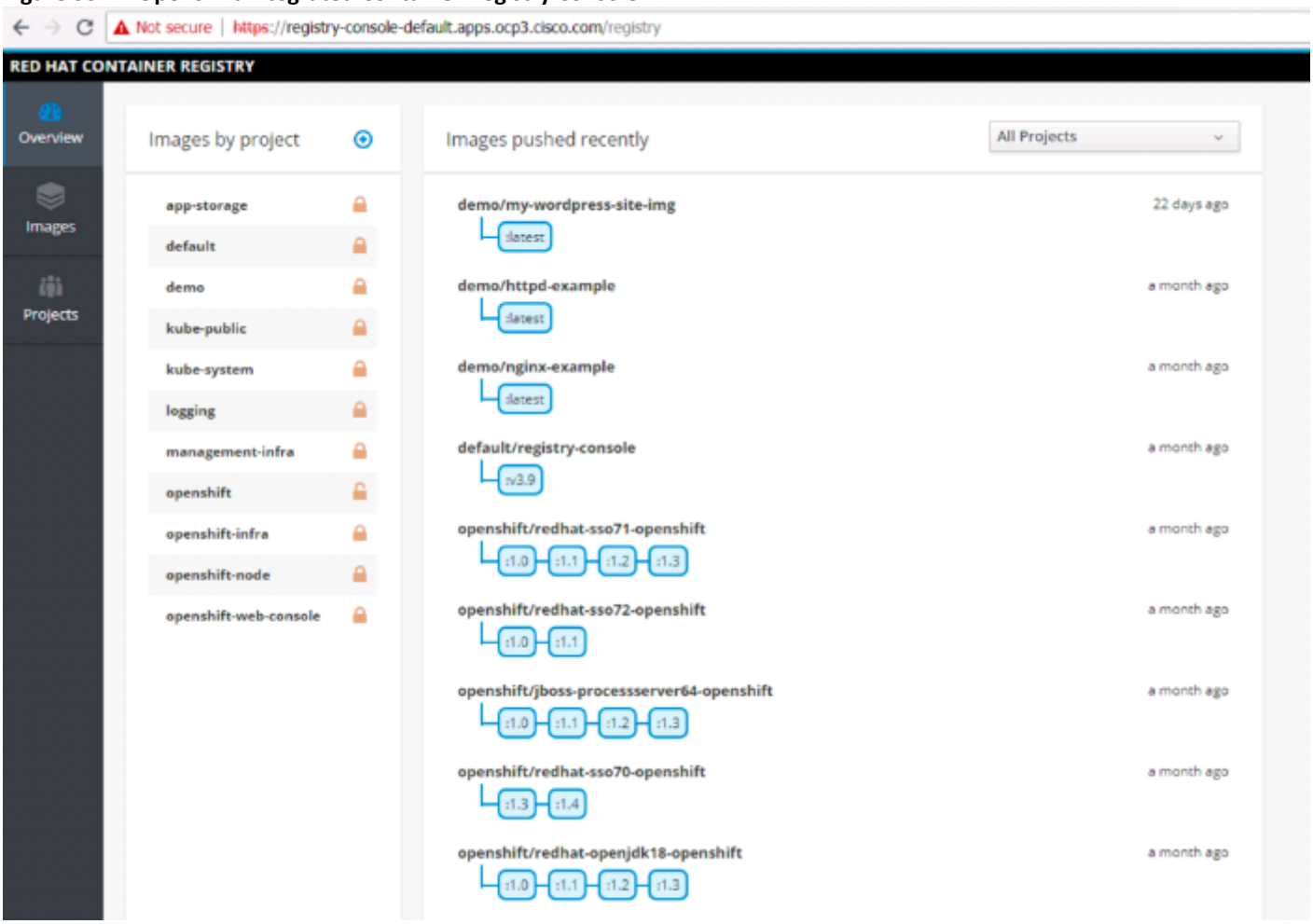
5. Login by providing Username and Password.

Figure 29 OpenShift Container Platform – Browser Catalog



6. Verify for OpenShift Integrated Container Registry console access, by clicking the registry console URL as displayed on dashboard under default project:

Figure 30 OpenShift Integrated Container Registry Console

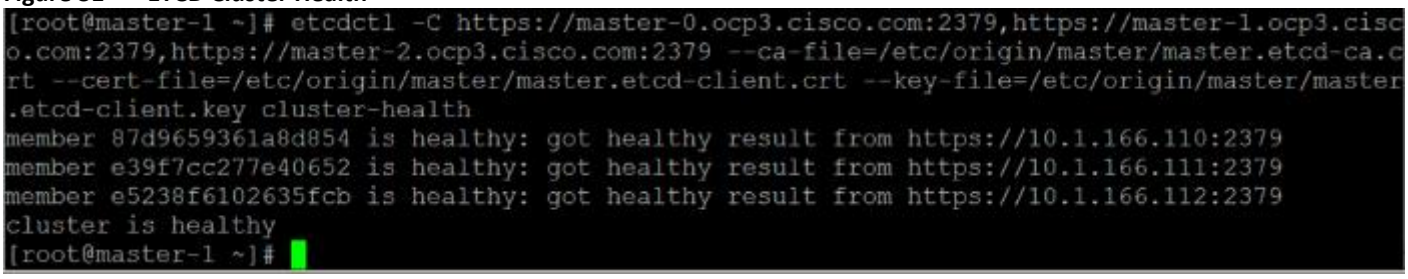


7. Verify the etcd cluster health and membership status. Login to one of the master node and run following commands:

```
 [root@master-1 ~]# etcdctl -C https://master-0.ocp3.cisco.com:2379,https://master-
1.ocp3.cisco.com:2379,https://master-2.ocp3.cisco.com:2379 --ca-
file=/etc/origin/master/master.etcd-ca.crt --cert-file=/etc/origin/master/master.etcd-
client.crt --key-file=/etc/origin/master/master.etcd-client.key cluster-health
```

Figure 31 shows the healthy status of the etcd cluster.

Figure 31 ETCD Cluster Health



8. Verify the member list by running the following command:

```
 [root@master-1 ~]# etcdctl -C https://master-0.ocp3.cisco.com:2379,https://master-
1.ocp3.cisco.com:2379,https://master-2.ocp3.cisco.com:2379 --ca-file=/etc/origin/master/master.etcd-
```

```
ca.crt --cert-file=/etc/origin/master/master.etcd-client.crt --key-file=/etc/origin/master/master.etcd-
client.key member list
```

Figure 32 shows the etcd member list and mentions which node is participating as a leader.

Figure 32 ETCD Member List

```
[root@master-1 ~]# etcdctl -C https://master-0.ocp3.cisco.com:2379,https://master-1.ocp3.cisc
o.com:2379,https://master-2.ocp3.cisco.com:2379 --ca-file=/etc/origin/master/master.etcd-ca.c
rt --cert-file=/etc/origin/master/master.etcd-client.crt --key-file=/etc/origin/master/master
.etcd-client.key member list
87d9659361a8d854: name=master-0.ocp3.cisco.com peerURLs=https://10.1.166.110:2380 clientURLs=
https://10.1.166.110:2379 isLeader=false
e39f7cc277e40652: name=master-1.ocp3.cisco.com peerURLs=https://10.1.166.111:2380 clientURLs=
https://10.1.166.111:2379 isLeader=true
e5238f6102635fcb: name=master-2.ocp3.cisco.com peerURLs=https://10.1.166.112:2380 clientURLs=
https://10.1.166.112:2379 isLeader=false
[root@master-1 ~]#
```

9. Follow these instructions to setup **OpenShift client** “oc” on the bastion instance and use this client to further validate the deployment:

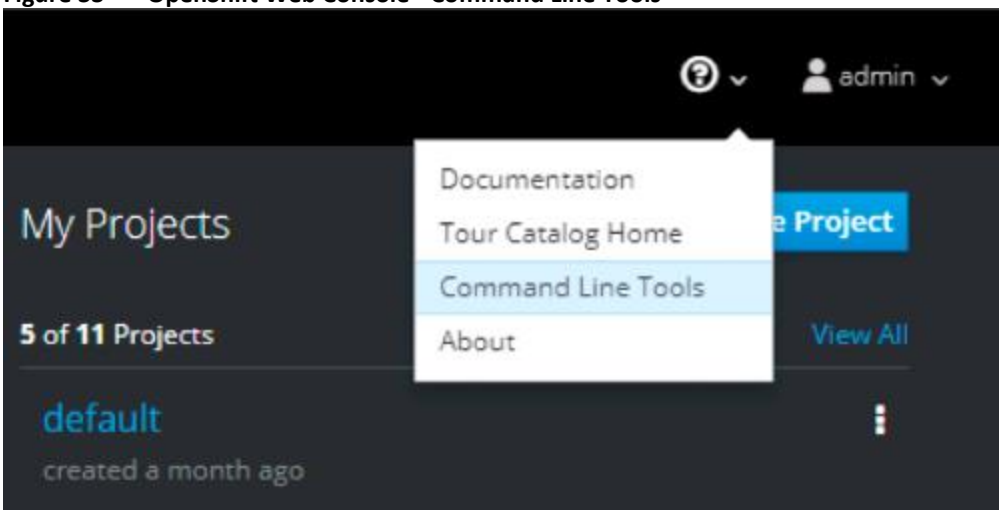
   a. Download and install OpenShift v3.9 Linux Client from this URL: https://access.redhat.com/downloads/content/290

   b. Once downloaded extract the .tar file and also copy it in /usr/bin and /usr/sbin folder as shown below.

```
[root@bastion oc]# tar -xvf oc-3.9.33-linux.tar.gz
[root@bastion oc]# ls
oc oc-3.9.33-linux.tar.gz
[root@bastion oc]# cp oc /usr/bin/.
[root@bastion oc]# cp oc /usr/sbin/.
```

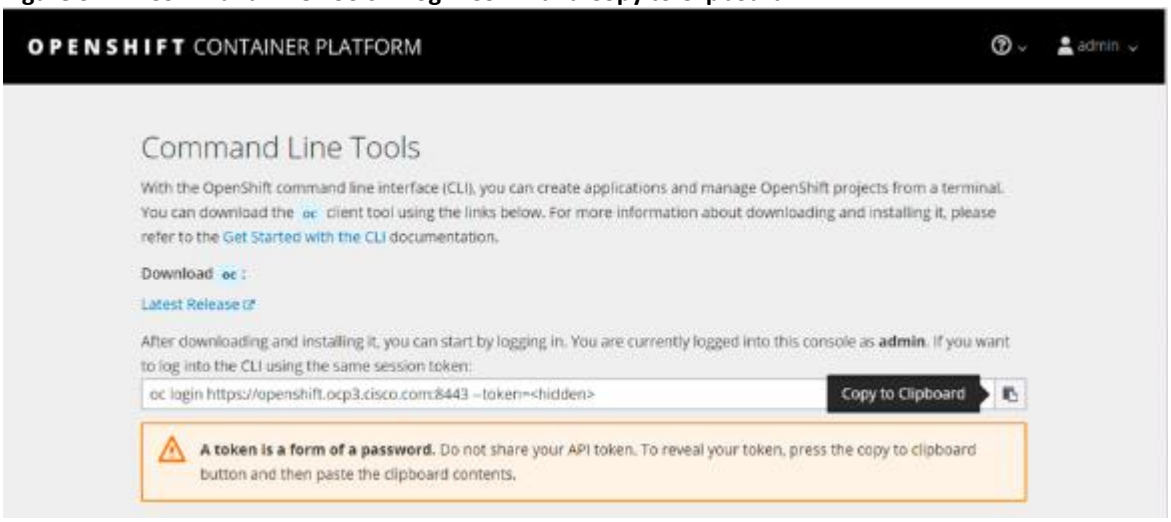
   c. Access OpenShift Web Console and click `?` on top right corner to get the authentication token as shown below.

Figure 33 OpenShift Web Console - Command Line Tools



   d. Copy login command with the token from the screenshot shown below:

Figure 34 Command Line Tools – Login Command Copy to Clipboard



e. Use copied command on the Bastion node to login to OpenShift Container cluster:

```
[root@bastion oc]# oc login https://openshift.ocp3.cisco.com:8443 --
token=XRmWO_zw2eJ9zet7UG25OgMRfZLm_kxuFG7jzZFraMs
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be intercepted by others.
Use insecure connections? (y/n): y

Logged into "https://openshift.ocp3.cisco.com:8443" as "admin" using the token provided.

You have access to the following projects and can switch between them with 'oc project <projectname>':

    app-storage
  * default
    demo
    kube-public
    kube-system
    logging
    management-infra
    openshift
    openshift-infra
    openshift-node
    openshift-web-console

Using project "default".
Welcome! See 'oc help' to get started.
```

f. Login through cli client `oc` can be done without having to supply authentication token by using `in-secure` connection as below. Here we need to provide username and password to login.

```
[root@bastion oc]# oc login https://openshift.ocp3.cisco.com:8443 --insecure-skip-tls-verify=true
Authentication required for https://openshift.ocp3.cisco.com:8443 (openshift)
Username: admin
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc project <projectname>':

    app-storage
  * default
    demo
    kube-public
    kube-system
    logging
    management-infra
    openshift
    openshift-infra
```

```
    openshift-node
    openshift-web-console

Using project "default".
[root@bastion oc]#
```

g. Verify container native storage PODs and other resources are running:

```
 [root@bastion oc]# oc project app-storage
[root@bastion oc]# oc get all
```

```
[root@bastion oc]# oc project app-storage
Now using project "app-storage" on server "https://openshift.ocp3.cisco.com:8443".
[root@bastion oc]# oc get all
NAME                  DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR
        AGE
ds/glusterfs-storage  3         3         3         3            3           glusterfs=stora
ge-host   27d

NAME                                REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfigs/heketi-storage    1          1         1         config

NAME                     HOST/PORT                                       PATH      SERVICES
      PORT      TERMINATION   WILDCARD
routes/heketi-storage    heketi-storage-app-storage.apps.ocp3.cisco.com             heketi-sto
rage   <all>                 None

NAME                          READY     STATUS    RESTARTS   AGE
po/glusterfs-storage-5hszr    1/1       Running   0          27d
po/glusterfs-storage-7nqld    1/1       Running   2          27d
po/glusterfs-storage-zvrpc    1/1       Running   0          27d
po/heketi-storage-1-fqm2s     1/1       Running   1          27d

NAME                  DESIRED   CURRENT   READY     AGE
rc/heketi-storage-1   1         1         1         27d

NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
svc/heketi-db-storage-endpoints     ClusterIP   172.30.126.234   <none>        1/TCP       27d
svc/heketi-storage                  ClusterIP   172.30.137.254   <none>        8080/TCP    27d
[root@bastion oc]#
```

h. Verify Gluster cluster status and storage resource health by logging in to one of the glusterfs-storage pods:

```
[root@bastion oc]# oc rsh po/glusterfs-storage-5hszr
sh-4.2# gluster peer status
Number of Peers: 2

Hostname: 10.1.166.120
Uuid: a872255e-d85a-441c-aa90-2bc64887fdbe
State: Peer in Cluster (Connected)

Hostname: 10.1.166.122
Uuid: eec34aa6-5dbf-49e9-94ce-14f380e548e7
State: Peer in Cluster (Connected)
sh-4.2#
sh-4.2# gluster volume list
glusterfs-registry-volume
heketidbstorage
vol_1487d27ac9805062124758c1085f71c3
vol_58380614fe1181e20ade3a0c3b5e4783
vol_617f6754b3520b36740147c568c5eae1
vol_a5b2f2168b97888ed26ef584fa544698
vol_c6608b88db1ca2e15d4c7593b37474fd
```

```
sh-4.2#
sh-4.2# gluster pool list
UUID                                    Hostname        State
a872255e-d85a-441c-aa90-2bc64887fdbe    10.1.166.120    Connected
eec34aa6-5dbf-49e9-94ce-14f380e548e7    10.1.166.122    Connected
2a4c3e83-89b3-43c9-bc68-5dd95326bb45    localhost       Connected
sh-4.2#
```

i. Verify Router and Registry Health, value in the DESIRED and CURRENT field should match:

```
[root@bastion oc]# oc -n default get deploymentconfigs/router
NAME      REVISION   DESIRED   CURRENT   TRIGGERED BY
router    1          3         3         config
[root@bastion oc]#
[root@bastion oc]# oc -n default get deploymentconfigs/docker-registry
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry   1          1         1         config
[root@bastion oc]#
```

j. Verify PODs are distributed on desired Infra nodes:

```
[root@bastion oc]# oc -n default get pods -o wide
NAME                       READY     STATUS    RESTARTS   AGE       IP              NODE
docker-registry-1-z8w57    1/1       Running   0          27d       172.16.4.9      infra-1.ocp3.cisco.com
registry-console-1-fs8qt   1/1       Running   0          27d       172.16.2.7      master-
0.ocp3.cisco.com
router-1-7kqxv             1/1       Running   0          27d       10.1.166.114    infra-1.ocp3.cisco.com
router-1-chtpg             1/1       Running   0          27d       10.1.166.113    infra-0.ocp3.cisco.com
router-1-nrtff             1/1       Running   0          27d       10.1.166.115    infra-2.ocp3.cisco.com
[root@bastion oc]#
```

Sample Application Test Scenario.

In this section, you will deploy a sample application to validate the complete environmental health of the deployment.

With the OpenShift “oc” client, complete the following steps:

1. Create a new project named “sample-test” as shown below:

```
[root@bastion oc]# oc new-project sample-test
Now using project "sample-test" on server "https://openshift.ocp3.cisco.com:8443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
[root@bastion oc]#
```

2. Deploy a sample application `cakephp-mysql-example`:

```
[root@bastion oc]# oc new-app cakephp-mysql-example
--> Deploying template "openshift/cakephp-mysql-example" to project sample-test

     CakePHP + MySQL (Ephemeral)
     ---------
     An example CakePHP application with a MySQL database. For more information about using this
template, including OpenShift considerations, see https://github.com/openshift/cakephp-
ex/blob/master/README.md.

     WARNING: Any data stored will be lost upon pod destruction. Only use this template for testing.

     The following service(s) have been created in your project: cakephp-mysql-example, mysql.
```

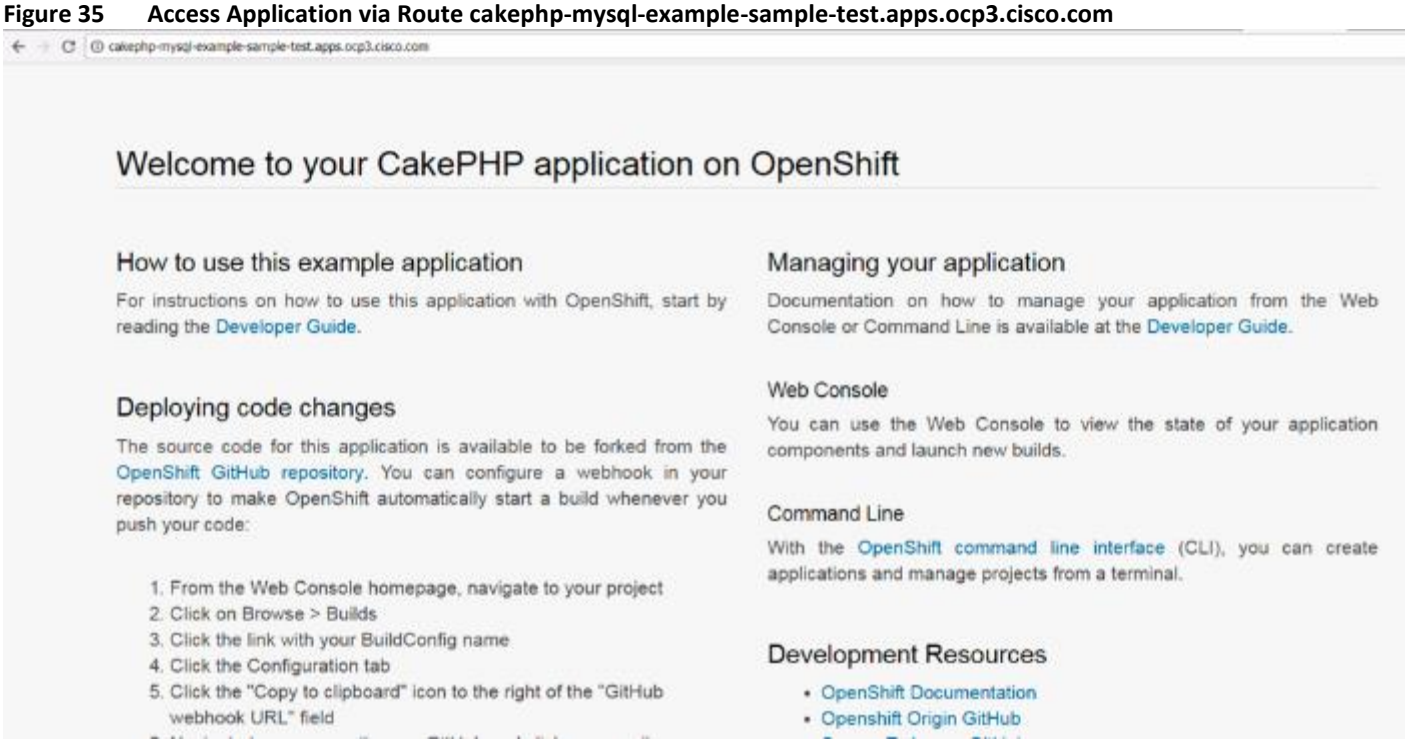
```
     For more information about using this template, including OpenShift considerations, see
https://github.com/openshift/cake-ex/blob/master/README.md.

     * With parameters:
        * Name=cakephp-mysql-example
        * Namespace=openshift
        * Memory Limit=512Mi
        * Memory Limit (MySQL)=512Mi
        * Git Repository URL=https://github.com/openshift/cakephp-ex.git
        * Git Reference=
        * Context Directory=
        * Application Hostname=
        * GitHub Webhook Secret=XdOl55T4ry4wG58hNrJINiBMXj6JH2yvKbFbvfhr # generated
        * Database Service Name=mysql
        * Database Engine=mysql
        * Database Name=default
        * Database User=cakephp
        * Database Password=QuCR3aD5j3gDKXBv # generated
        * CakePHP secret token=Og_nU4fJH7WGKlB1vVLpS8R2kQxa4Wp3KHzza5PA0bCGqIDOkp # generated
        * CakePHP Security Salt=tfh52qgntPPyHCoAmHh7qbycT3XD3nH2TnfWCyMM # generated
        * CakePHP Security Cipher Seed=656325452333584321308842208434 # generated
        * OPcache Revalidation Frequency=2
        * Custom Composer Mirror URL=

--> Creating resources ...
    secret "cakephp-mysql-example" created
    service "cakephp-mysql-example" created
    route "cakephp-mysql-example" created
    imagestream "cakephp-mysql-example" created
    buildconfig "cakephp-mysql-example" created
    deploymentconfig "cakephp-mysql-example" created
    service "mysql" created
    deploymentconfig "mysql" created
--> Success
    Access your application via route 'cakephp-mysql-example-sample-test.apps.ocp3.cisco.com'
    Build scheduled, use 'oc logs -f bc/cakephp-mysql-example' to track its progress.
    Run 'oc status' to view your app.
[root@bastion oc]#
```

3. After the successful build, we should see two POD running i.e. application POD and database POD as shown below.

```
[root@bastion oc]# oc get pods -o wide
NAME                            READY     STATUS      RESTARTS   AGE       IP             NODE
cakephp-mysql-example-1-build   0/1       Completed   0          5m        172.16.20.18   storage-
02.ocp3.      cisco.com
cakephp-mysql-example-1-x74gc   1/1       Running     0          4m        172.16.10.18   storage-
01.ocp3.      cisco.com
mysql-1-pftpx                   1/1       Running     0          5m        172.16.10.17   storage-
01.ocp3.      cisco.com
[root@bastion oc]#
```

4. Verify to access the application via route 'cakephp-mysql-example-sample-test.apps.ocp3.cisco.com' by copy and paste in the browser as shown below.

Figure 35 Access Application via Route cakephp-mysql-example-sample-test.apps.ocp3.cisco.com

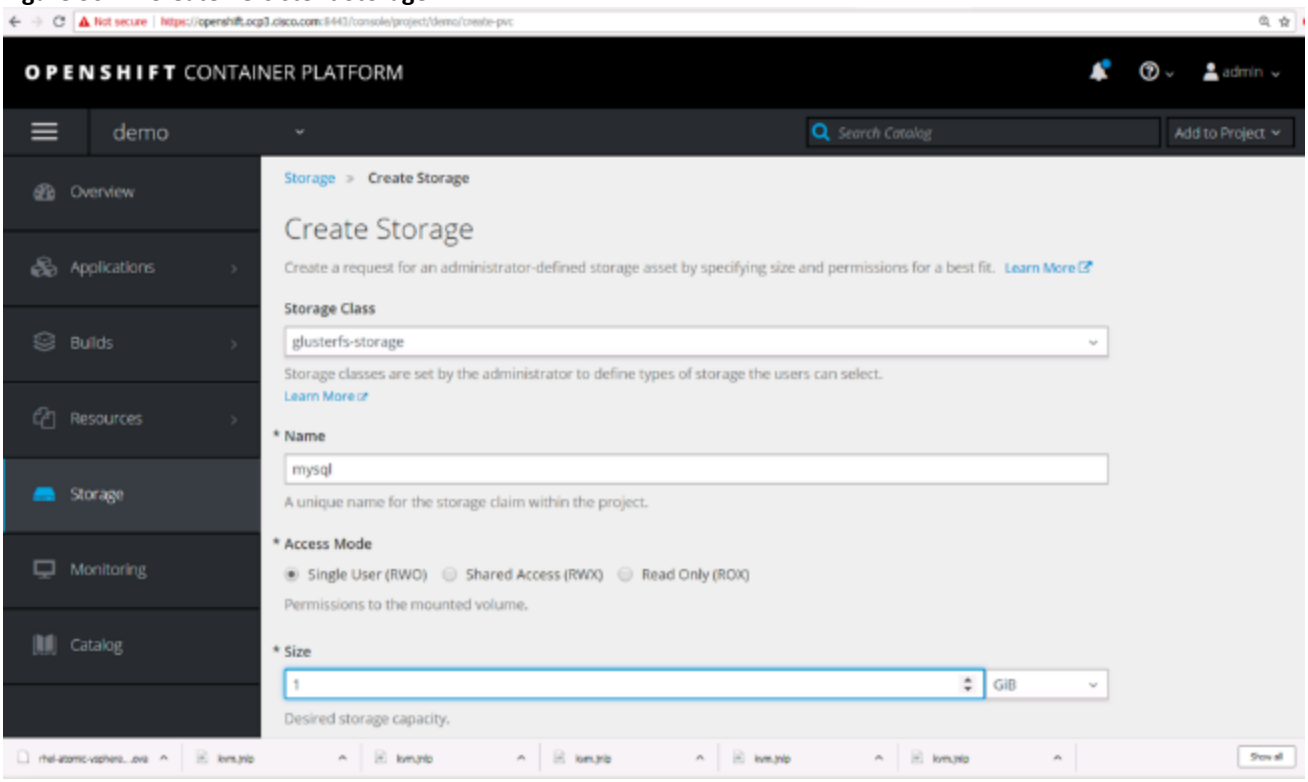


Web Console – UI Operations

To create a project and deploy a sample application from the service catalog, complete the following steps:

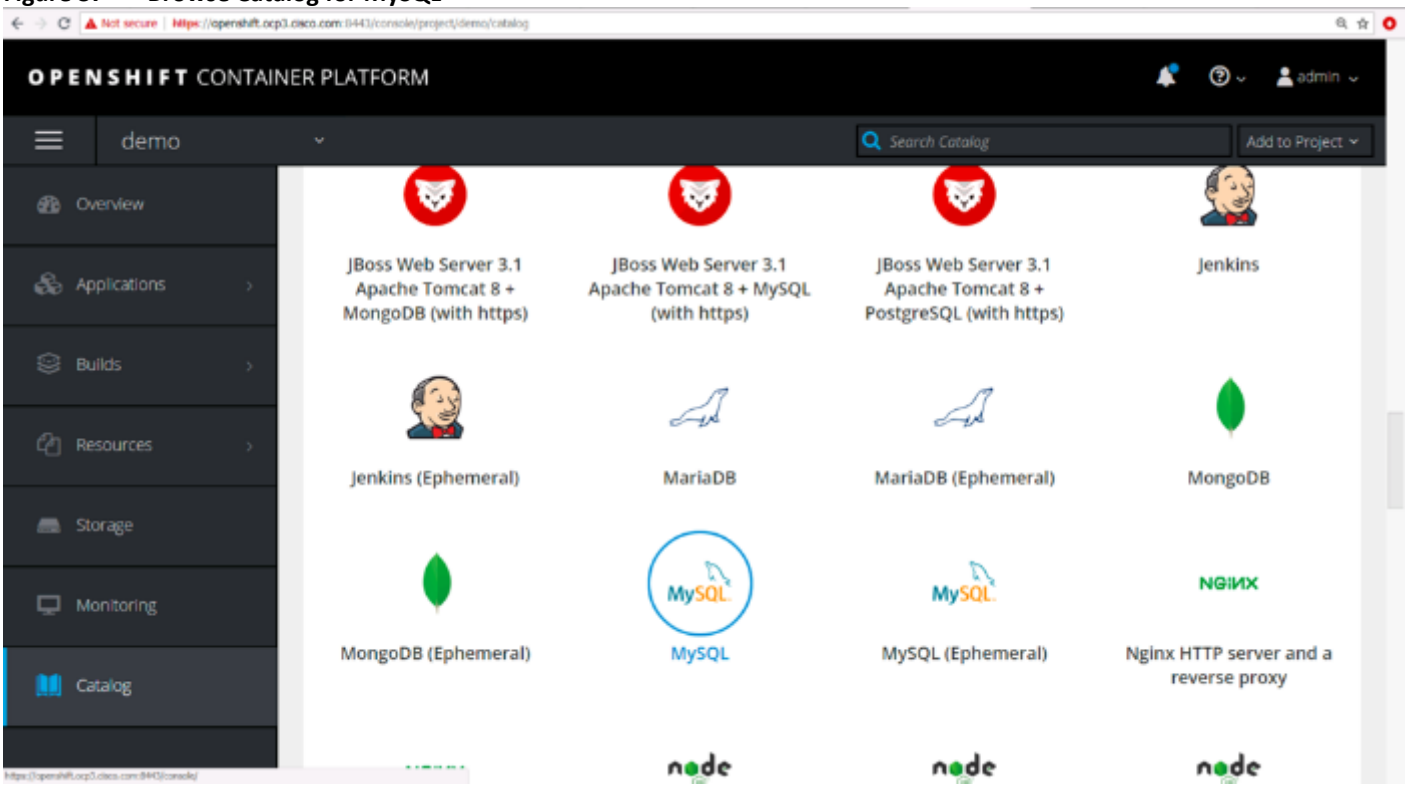
1. Select previously created project or create a new one. For example demo project under project-list
2. Create storage by clicking Storage in the left pane and Create Storage button on the right pane. Select Storage Class, name of the PV claim, and size as shown below:

Figure 36 Create Persistent Storage



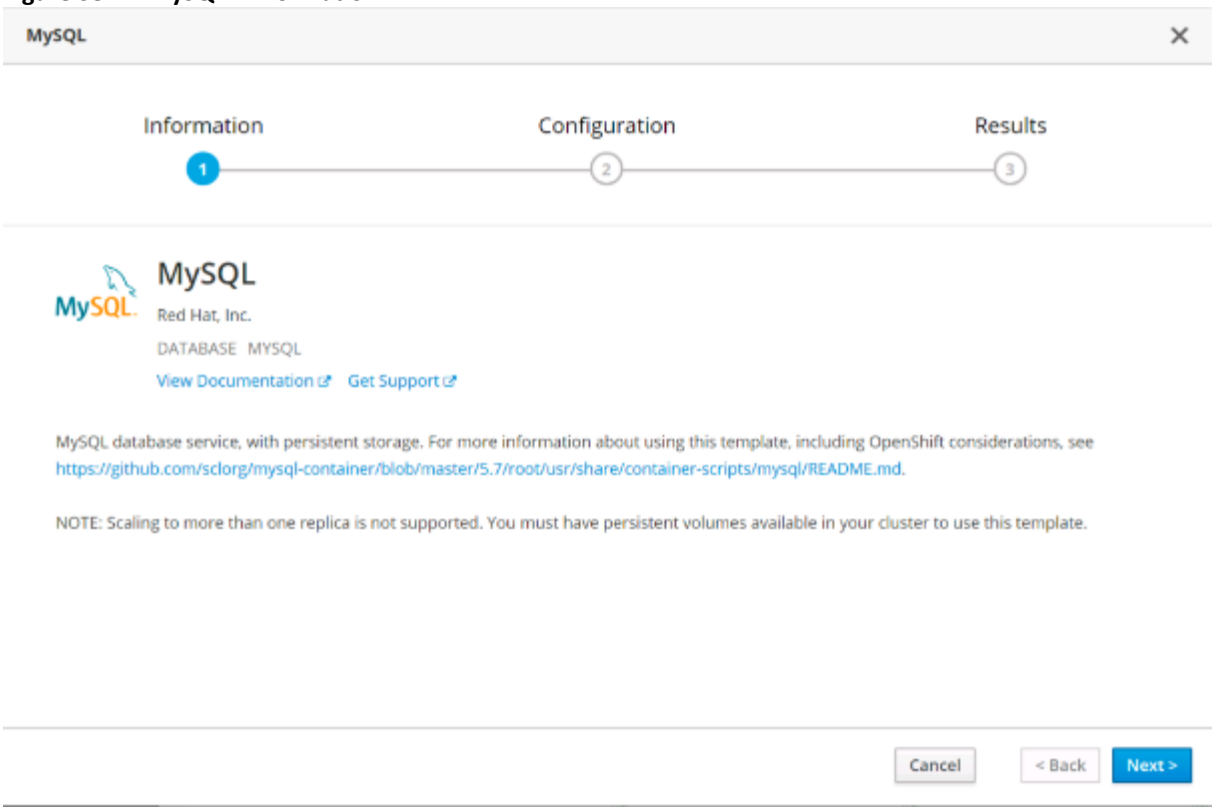
3. Browse Catalog by clicking Catalog in the left pane for MySQL. Click MySQL.

Figure 37 Browse Catalog for MySQL



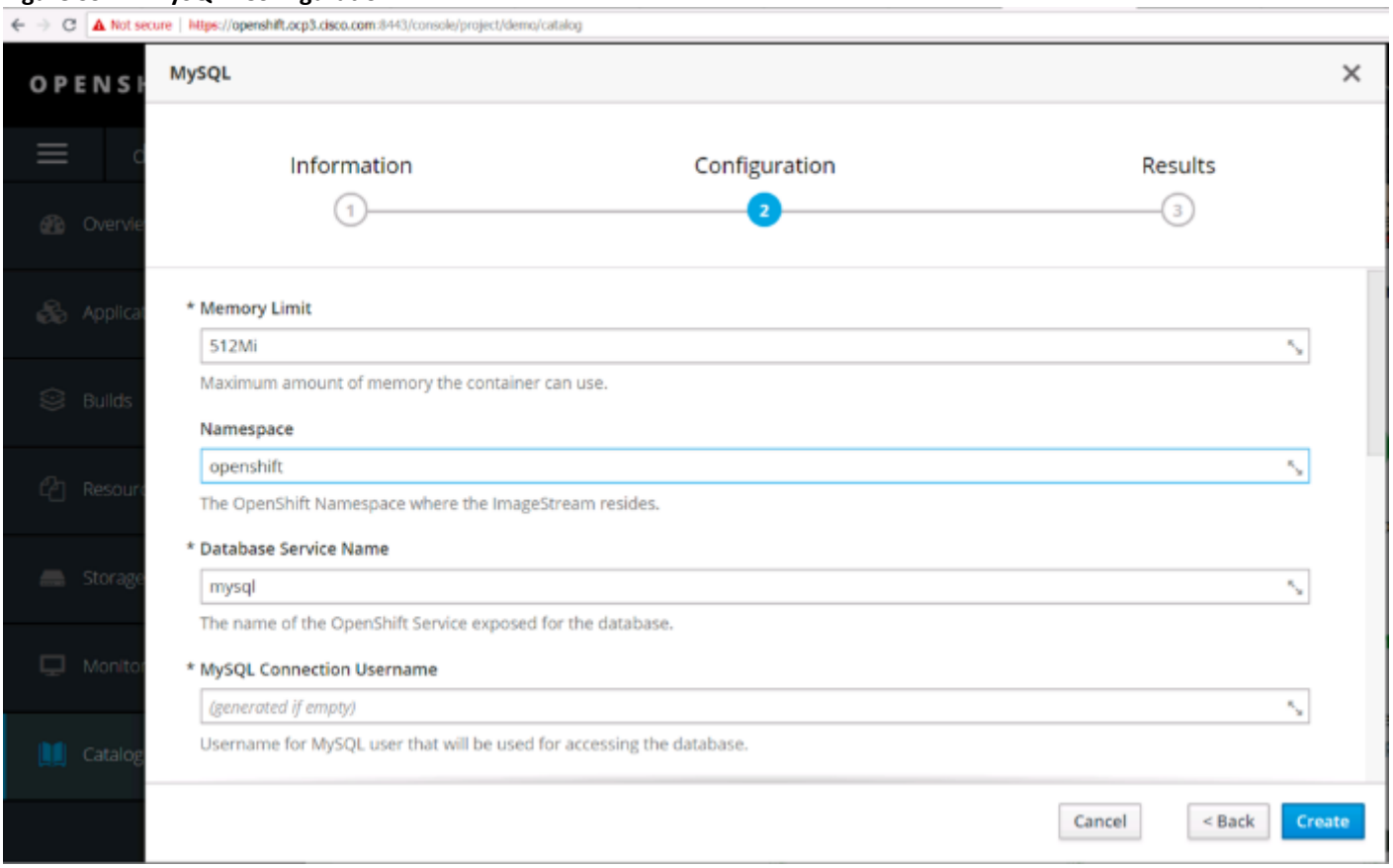
4. MySQL Information screen will show up as shown below. Click Next

Figure 38 MySQL - Information



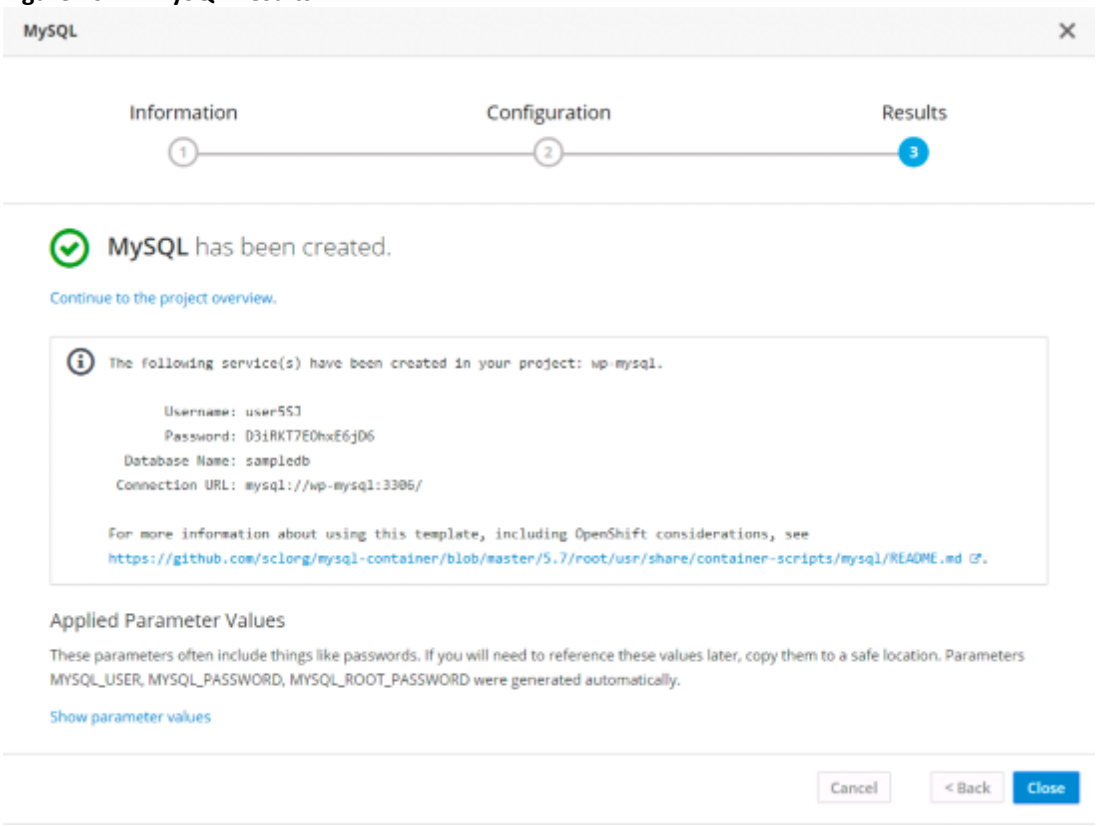
5. Specify database service name for example mysql in this case. Click Create

Figure 39 MySQL - Configuration



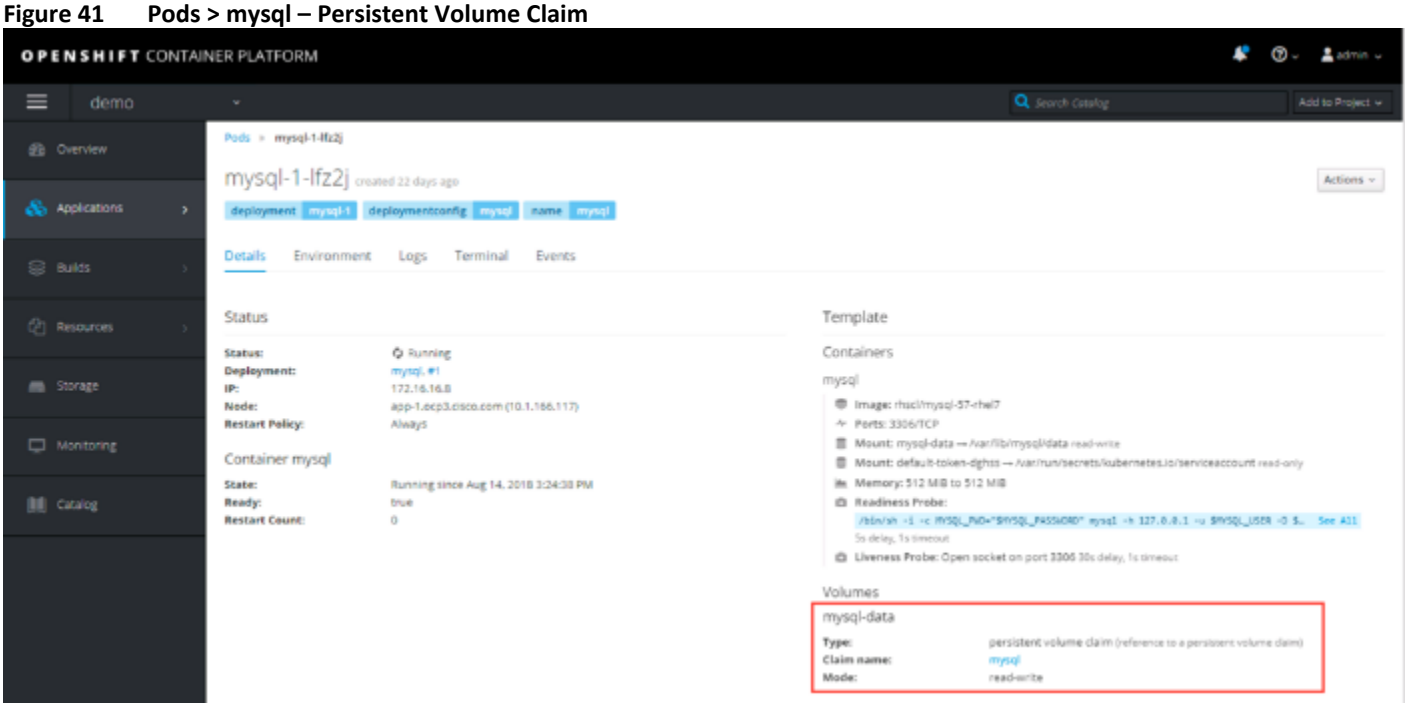
6. Click Close.

Figure 40 MySQL-Results



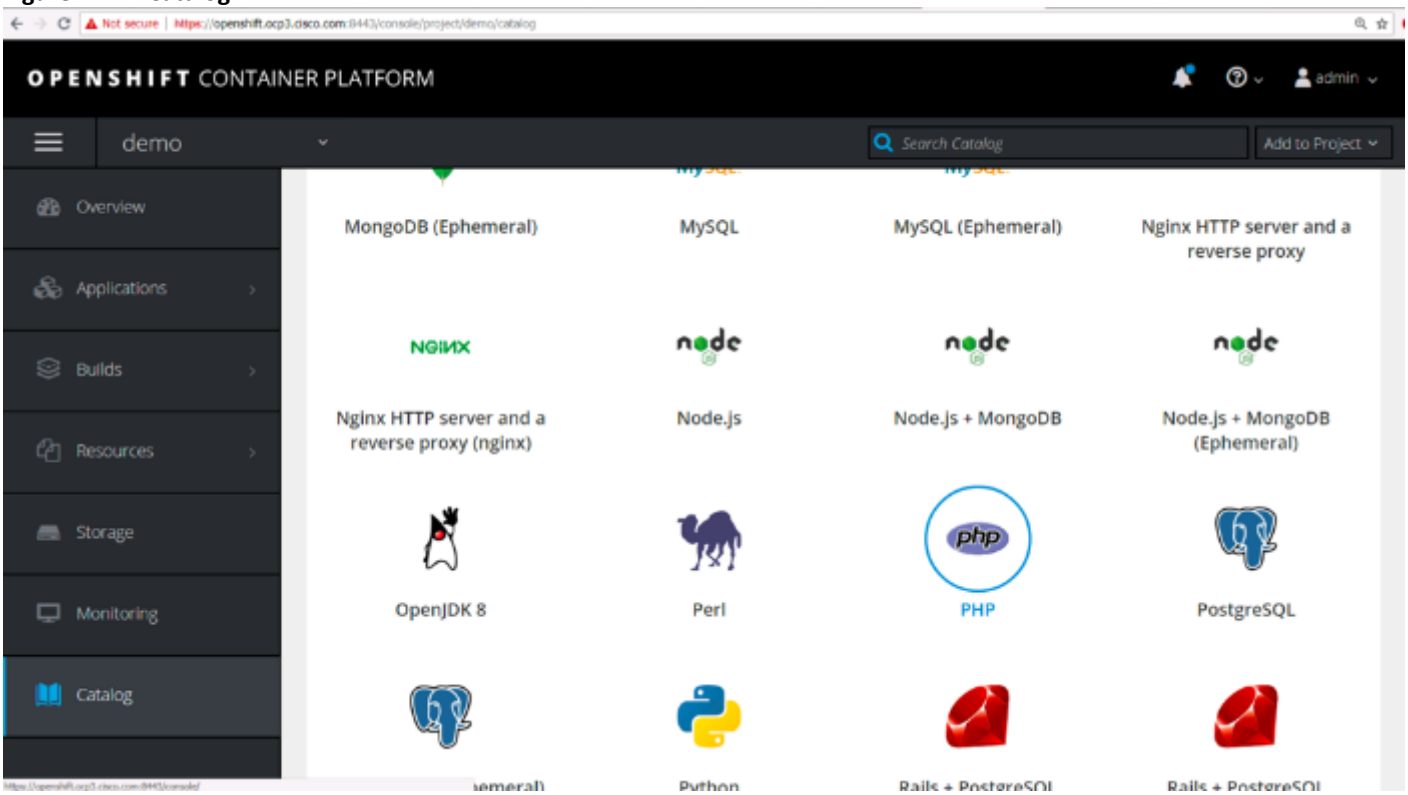
7. Verify persistent volume claim in mysql pod.

Figure 41 Pods > mysql – Persistent Volume Claim



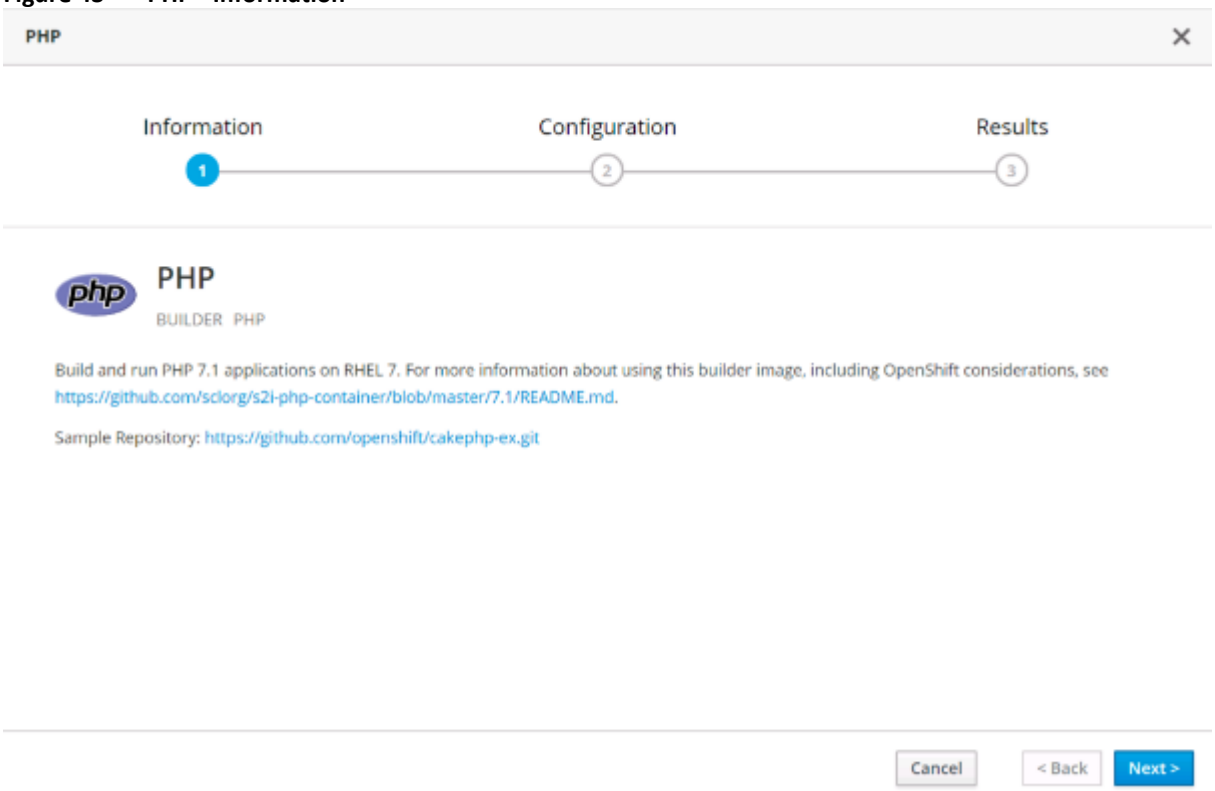
8. Browse Catalog for PHP application. Click PHP.

Figure 42 Catalog - PHP



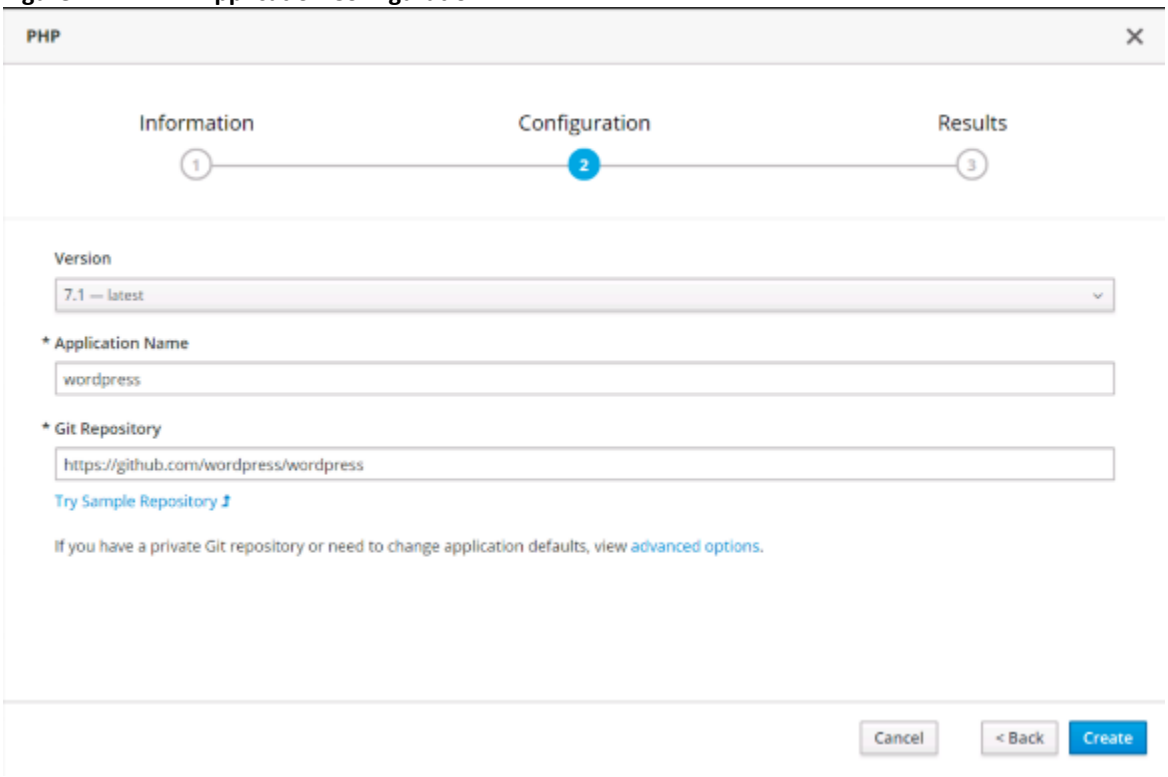
9. Click Next on PHP information screen as shown below

Figure 43 PHP - Information



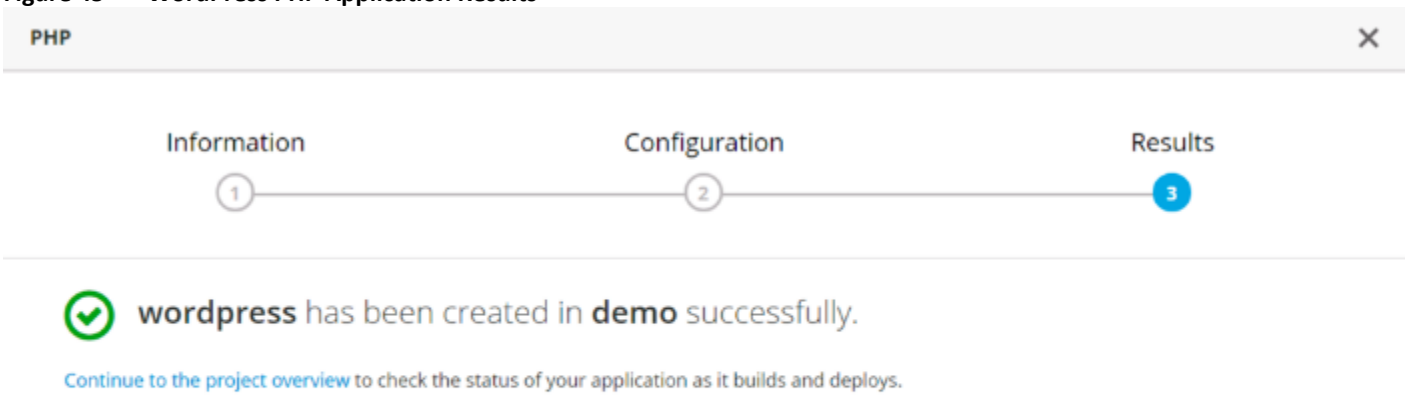
10. Specify the PHP application name, for example wordpress, in this case and git repository as https://github.com/wordpress/wordpress. Click Create.

Figure 44 PHP Application Configuration



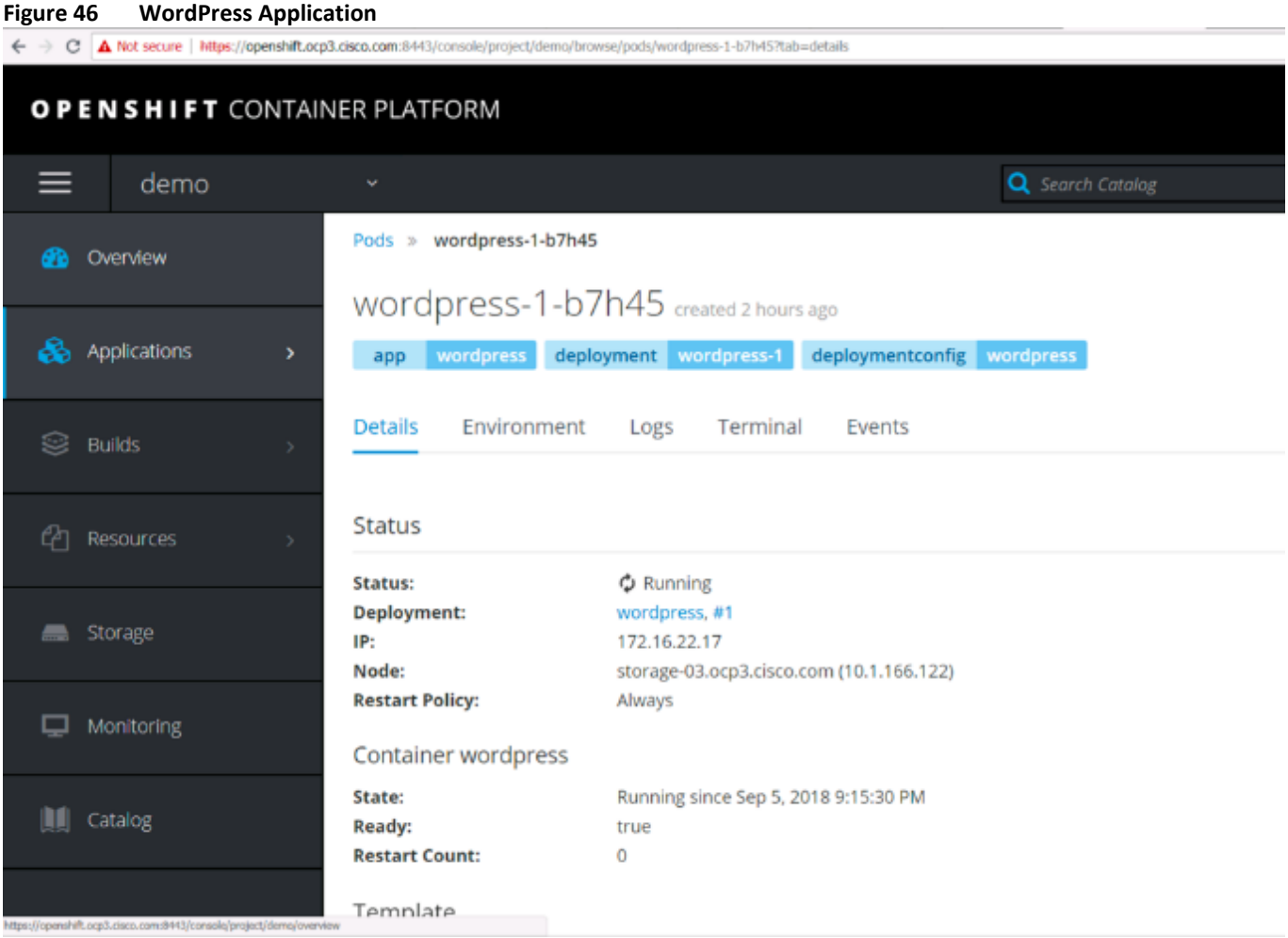
11. Wordpress PHP application is created as show below. Click Close.

Figure 45 WordPress PHP Application Results



12. Verify the Build process is completed as status shows “Running” as shown below.

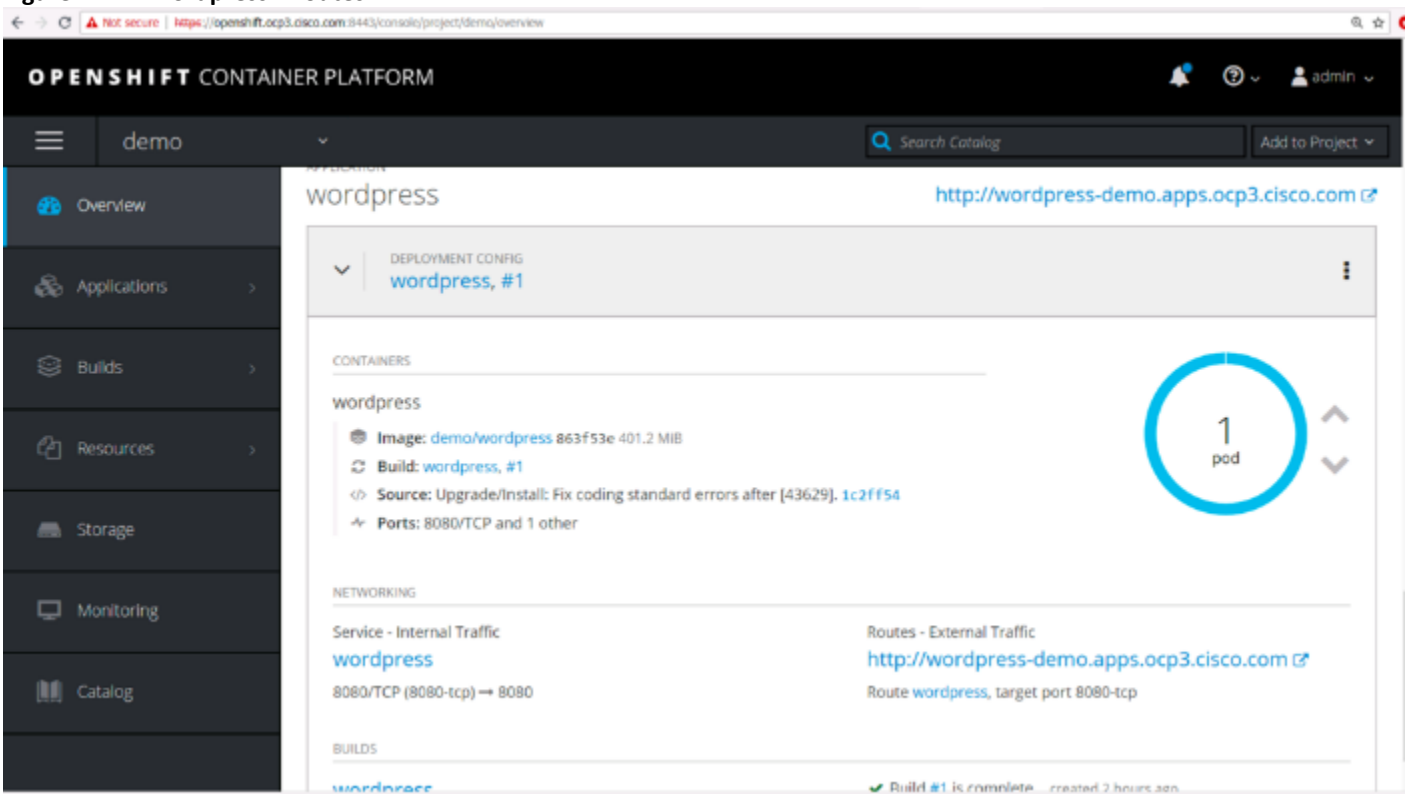
Figure 46 WordPress Application



As shown in Figure 46, the PHP pod is provisioned in storage node, such as storage-03.ocp3.cisco.com. As mentioned previously in this document, Cisco C240 M5 nodes are not only providing GlusterFS, it is also acting as compute nodes to provision pods. In this reference architecture, pods can be provisioned in virtualized nodes as well as in bare-metal nodes.

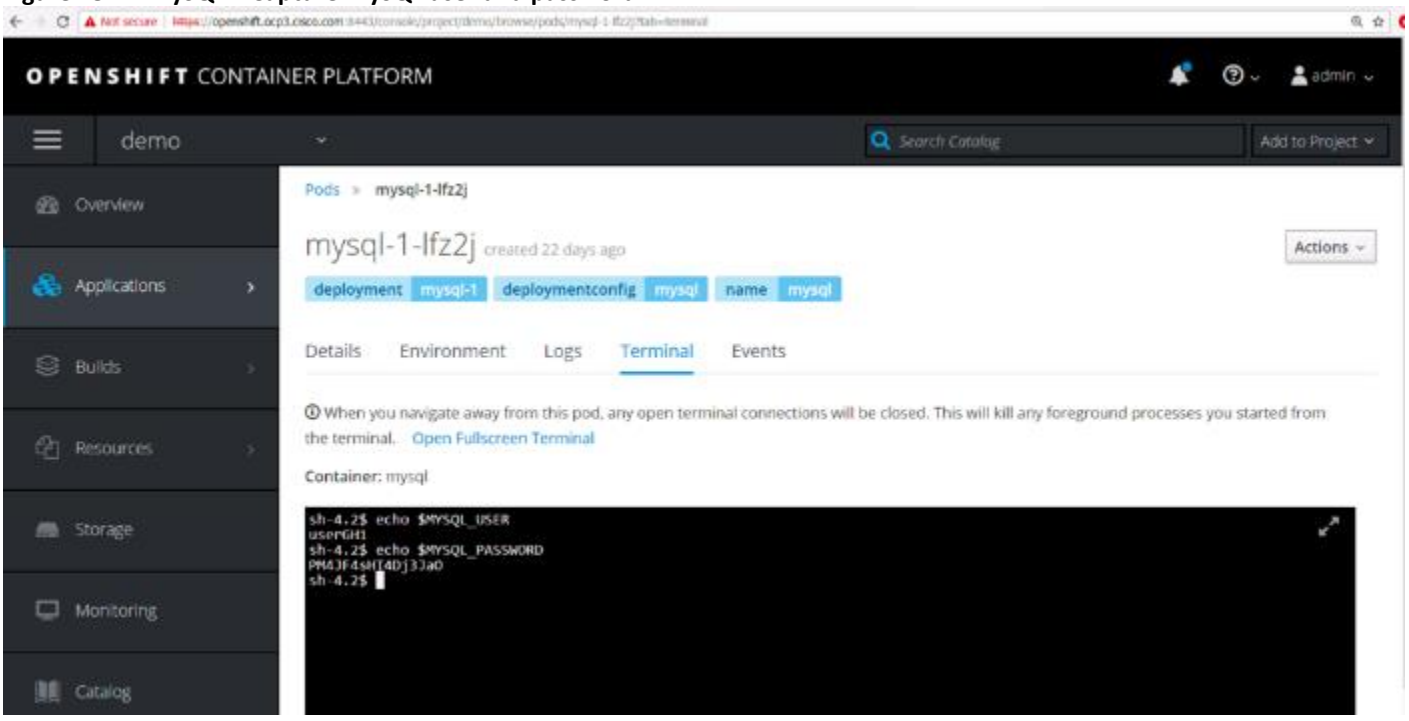
13. Launch wordpress by clicking the Routes that is exposing this service as shown below:

Figure 47 Wordpress - Routes



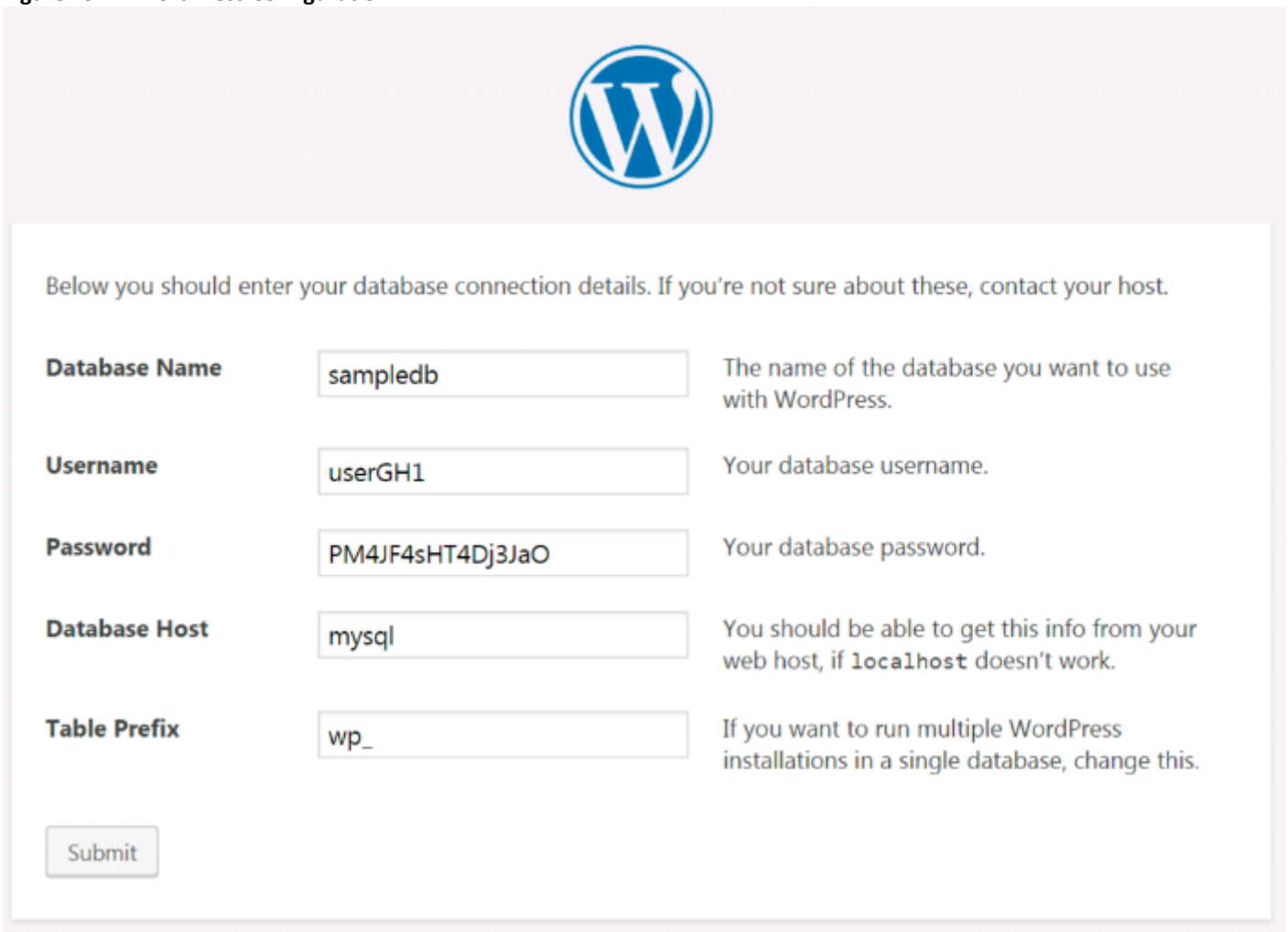
14. Go to the Terminal of mysql to capture the mysql user and password that would be needed to configure and install wordpress. This information is also available in the Results screen while create mysql pod.

Figure 48 MySQL – Capture MySQL user and password



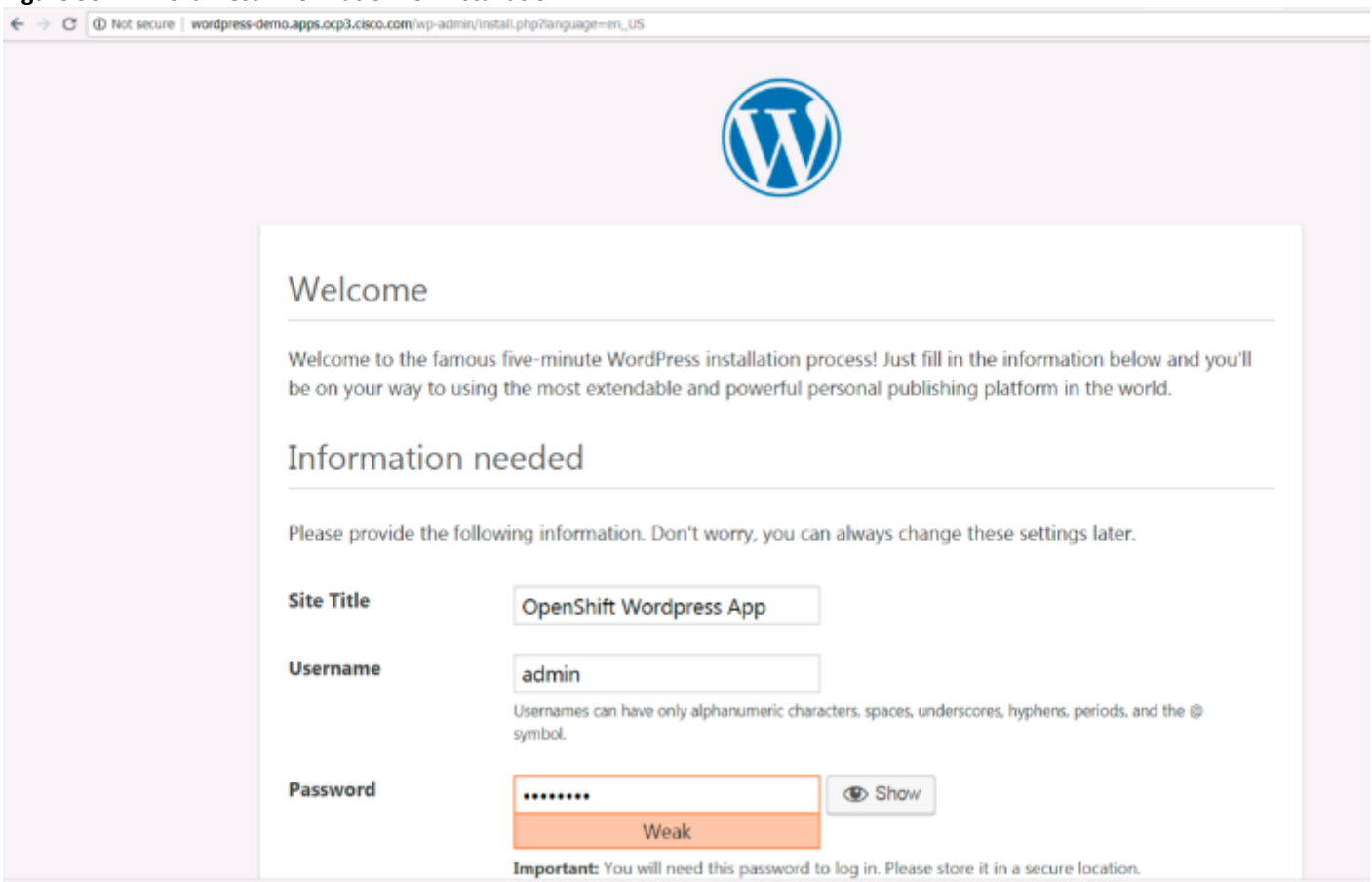
15. Configure wordpress for installation as shown below:

Figure 49 WordPress Configuration



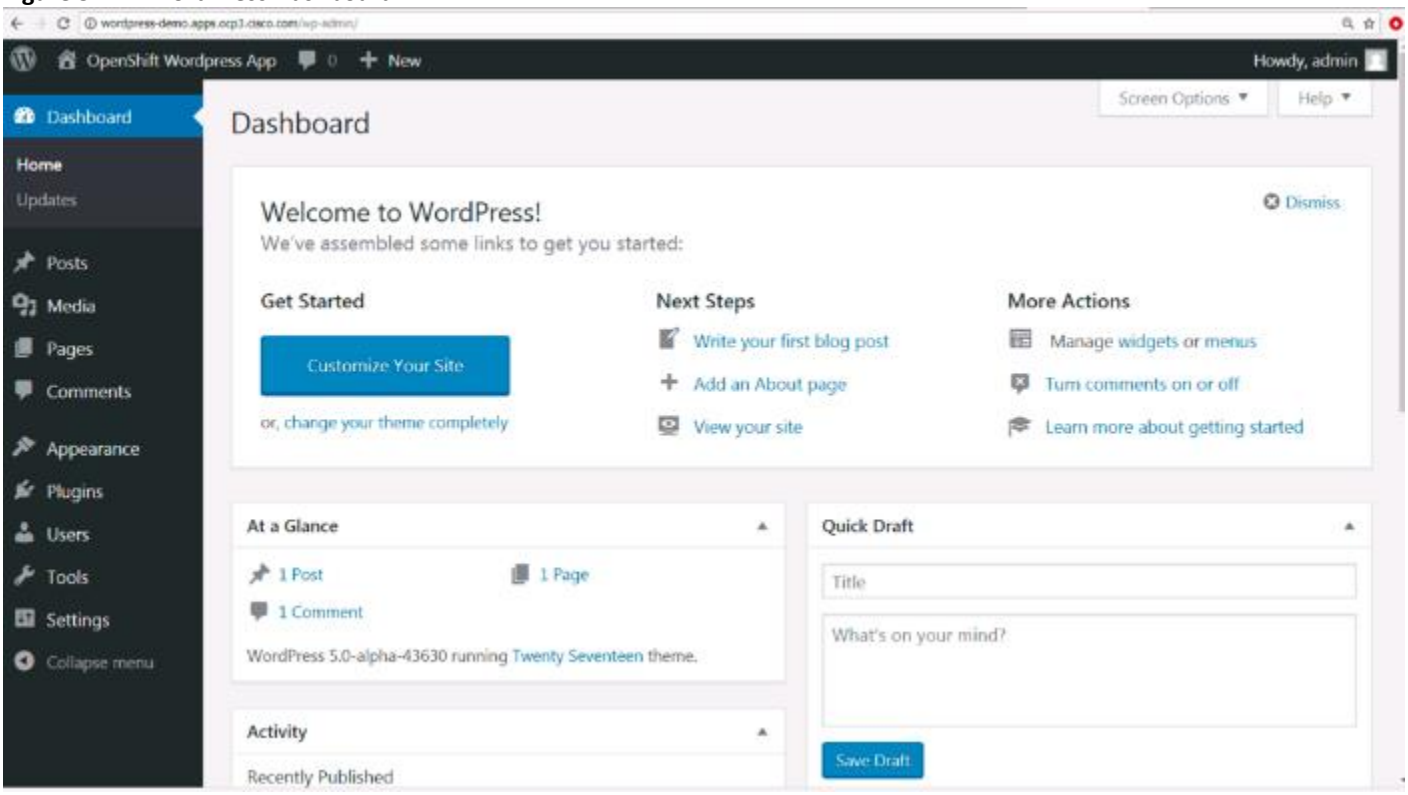
16. Install wordpress by providing all required fields such as Site Title, Username, Password, and email address.

Figure 50 WordPress Information for Installation



17. You will receive a successful installation notice and you are ready to login. Click Login and provide the credentials used in setting up the installation in step 15. Figure 51 shows the WordPress Dashboard.

Figure 51 WordPress Dashboard



18. On CLI using OpenShift Client "OC", get the application pod and pvc status verified as shown below:

```
[root@bastion oc]# oc project demo
[root@bastion oc]# oc get pods -o wide|grep wordpress
wordpress-1-b7h45               1/1       Running             0          2h        172.16.22.17   storage-
03.ocp3.cisco.com
wordpress-1-build               0/1       Completed           0          2h        172.16.16.18   app-
1.ocp3.cisco.com
[root@bastion oc]#
```

```
[root@bastion oc]# oc get pvc|grep mysql
mysql                           Bound         pvc-a969adab-9ff5-11e8-86cc-0050568a8862   1Gi
RWO            glusterfs-storage   22m
[root@bastion oc]#
```

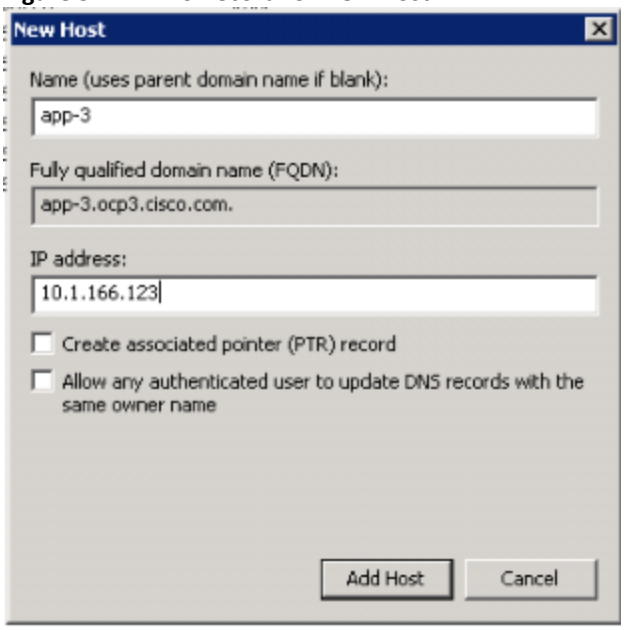
```
[root@bastion oc]# oc get pv  | grep mysql
pvc-a969adab-9ff5-11e8-86cc-0050568a8862   1Gi        RWO            Delete           Bound
demo/mysql                              glusterfs-storage            22m
[root@bastion oc]#
```

Scale the Environment

To add more master or app/infra nodes in the cluster, complete the following steps:

1. Add DNS record for the newly added node.

Figure 52 DNS Record for New Host



2. Create an inventory file for adding new nodes to be used by prod.yaml. This step provision VM instances for new hosts. You can scale to more than one node at the same time by specifying the nodes. This file can be created by copying the /etc/ansible/hosts file and update accordingly as shown in the below example. Important entries have been highlighted in bold.

```
 [root@bastion ansible]# cat add-app-node
[OSEv3:children]
ansible
masters
infras
apps
etcd
nodes
lb
glusterfs
glusterfs_registry

[OSEv3:vars]
ansible_ssh_user=root
deployment_type=openshift-enterprise
debug_level=2
openshift_release="3.9"
openshift_enable_service_catalog=false

# See https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
#openshift_examples_modify_imagestreams=true
openshift_disable_check=docker_image_availability

console_port=8443
openshift_debug_level="{{ debug_level }}"

openshift_node_debug_level="{{ node_debug_level | default(debug_level,true) }}"
openshift_master_debug_level="{{ master_debug_level |default(debug_level, true) }}"


openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge': 'true',
'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]

openshift_hosted_router_replicas=3
openshift_master_cluster_method=native
openshift_enable_service_catalog=false
```

```
osm_cluster_network_cidr=172.16.0.0/16
openshift_node_local_quota_per_fsgroup=512Mi

openshift_cloudprovider_vsphere_username="administrator@vsphere.local"
openshift_cloudprovider_vsphere_password="H1ghV0lt!"
openshift_cloudprovider_vsphere_host="ocp-vcenter.aflexpod.cisco.com"
openshift_cloudprovider_vsphere_datacenter=OCP-Datacenter
openshift_cloudprovider_vsphere_cluster=OCP-Cluster
openshift_cloudprovider_vsphere_resource_pool=ocp39
openshift_cloudprovider_vsphere_datastore="datastore4"
openshift_cloudprovider_vsphere_folder="ocp39"
openshift_cloudprovider_vsphere_template="Rhel75"
openshift_cloudprovider_vsphere_vm_network="VM Network"
openshift_cloudprovider_vsphere_vm_netmask="255.255.255.0"
openshift_cloudprovider_vsphere_vm_gateway="10.1.166.1"
openshift_cloudprovider_vsphere_vm_dns="10.1.166.9"

default_subdomain=ocp3.cisco.com

load_balancer_hostname=openshift.ocp3.cisco.com
openshift_master_cluster_hostname="{{ load_balancer_hostname }}"
openshift_master_cluster_public_hostname="{{ load_balancer_hostname }}"
openshift_master_default_subdomain="apps.ocp3.cisco.com"

os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
osm_use_cockpit=false

openshift_clock_enabled=true

# CNS registry storage
openshift_hosted_registry_replicas=1
openshift_registry_selector="region=infra"
openshift_hosted_registry_storage_kind=glusterfs
openshift_hosted_registry_storage_volume_size=30Gi

# CNS storage cluster for applications
openshift_storage_glusterfs_namespace=app-storage
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_block_deploy=false

# CNS storage for OpenShift infrastructure
openshift_storage_glusterfs_registry_namespace=infra-storage
openshift_storage_glusterfs_registry_storageclass=false
openshift_storage_glusterfs_registry_block_deploy=true
openshift_storage_glusterfs_registry_block_storageclass=true
openshift_storage_glusterfs_registry_block_storageclass_default=true
openshift_storage_glusterfs_registry_block_host_vol_create=true
openshift_storage_glusterfs_registry_block_host_vol_size=100

# red hat subscription name and password
rhsub_user=<username>
rhsub_pass=<password>
rhsub_pool=<pool-id>

#registry
openshift_public_hostname=openshift.ocp3.cisco.com

[ansible]
localhost




[masters]

[infras]


[apps]
app-3 openshift_node_labels="{'region': 'app'}" ipv4addr=10.1.166.123

[etcd]
```

```
[lb]

[storage]


[nodes]
app-3.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-3.ocp3.cisco.com


[glusterfs]

[glusterfs_registry]

[root@bastion ansible]#
```

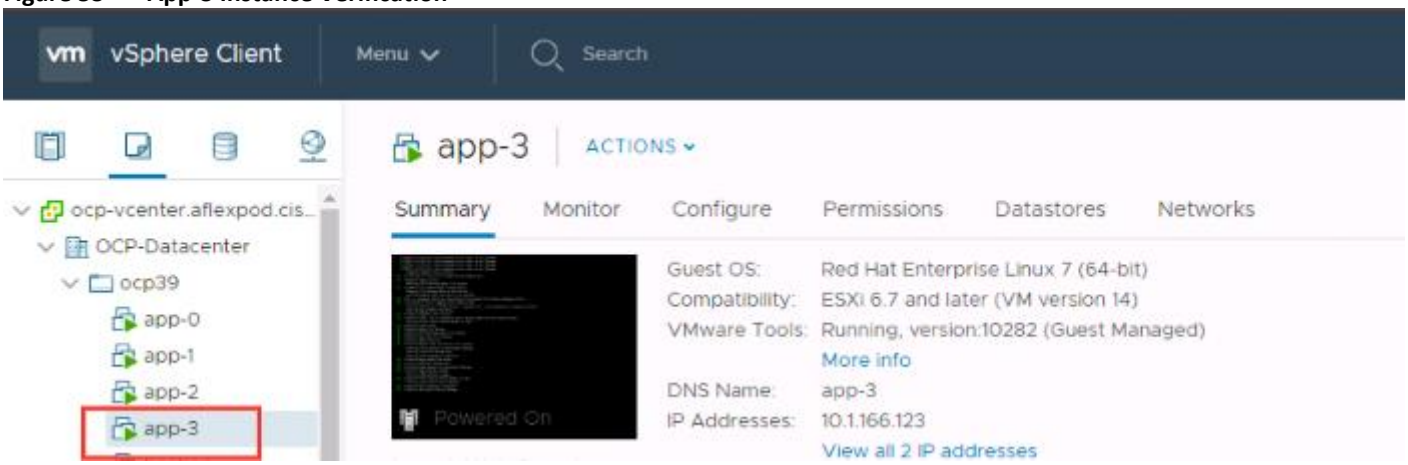
3. Run the prod.yaml file as shown below:

```
[root@bastion ansible]# ansible-playbook -i add-app-node openshift-ansible-contrib/reference-
architecture/vmware-ansible/playbooks/prod.yaml
```

If ssh key is not added in VM template via ssh-copy-id, playbook will ask you to type 'yes' while gathering facts for the authenticity of the newly created host. If prod.yaml playbook fails in establishing the authenticity of the host, launch the web console in vCenter and verify that IP address has been assigned and host name has been updated by running "**# ip a**" and "**# hostnamectl status**" command. Run **# ansible -i add-app-node all -m ping** to verify success and re-run the prod.yaml playbook.

4. Verify VM in vCenter.

Figure 53 App-3 Instance Verification



5. Edit your **/etc/ansible/hosts** file and add new**_<host_type>** to the **[OSEv3:children]** section:

```
 [OSEv3:children]
masters
nodes
new_nodes
```

6. To add new master hosts, add `new_masters`.
7. Create a `[new_<host_type>]` section much like an existing section, specifying host information for any new hosts you want to add. For example, when adding a new node:

```
 [nodes]
```

```
master-0.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-0.ocp3.cisco.com
master-1.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-1.ocp3.cisco.com
master-2.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-2.ocp3.cisco.com
infra-0.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
0.ocp3.cisco.com
infra-1.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
1.ocp3.cisco.com
infra-2.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
2.ocp3.cisco.com
app-0.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-0.ocp3.cisco.com
app-1.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-1.ocp3.cisco.com
app-2.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-2.ocp3.cisco.com
storage-01.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
01.ocp3.cisco.com
storage-02.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
02.ocp3.cisco.com
storage-03.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
03.ocp3.cisco.com

[new_nodes]
app-3.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-3.ocp3.cisco.com
```

8. When adding new masters, hosts added to the **[new_masters]** section must also be added to the **[new_nodes]** section. This helps ensure the new master host is part of the OpenShift SDN.

```
 [masters]
master-0.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.110
master-1.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.111
master-2.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.112

[new_masters]
master3.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" ipv4addr=10.1.166.124


[nodes]
master-0.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-0.ocp3.cisco.com
master-1.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-1.ocp3.cisco.com
master-2.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-2.ocp3.cisco.com
infra-0.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
0.ocp3.cisco.com
infra-1.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
1.ocp3.cisco.com
infra-2.ocp3.cisco.com openshift_node_labels="{'region': 'infra'}" openshift_hostname=infra-
2.ocp3.cisco.com
app-0.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-0.ocp3.cisco.com
app-1.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-1.ocp3.cisco.com
app-2.ocp3.cisco.com openshift_node_labels="{'region': 'app'}" openshift_hostname=app-2.ocp3.cisco.com
storage-01.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
01.ocp3.cisco.com
storage-02.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
02.ocp3.cisco.com
storage-03.ocp3.cisco.com  openshift_node_labels="{'region': 'storage'}" openshift_hostname=storage-
03.ocp3.cisco.com

[new_nodes]
master-3.ocp3.cisco.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=true
openshift_hostname=master-3.ocp3.cisco.com
```

9. Run the ***scaleup.yml*** playbook. If your inventory file is located somewhere other than the default of ***/etc/ansible/hosts***.

10. For additional nodes:

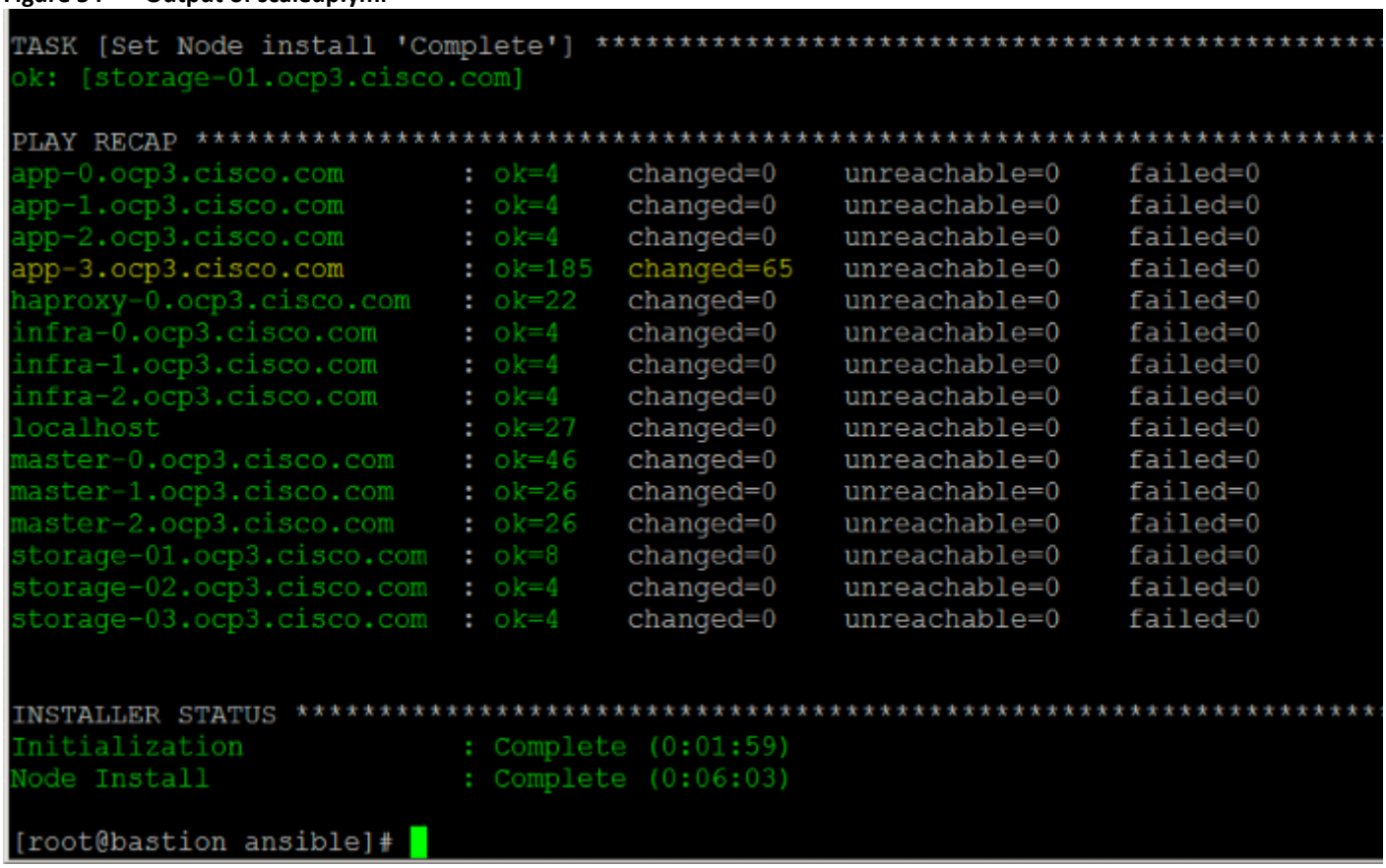
```
# ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-node/scaleup.yml
```

11. For additional masters:

```
# ansible-playbook /usr/share/ansible/openshift-ansible/playbooks/openshift-master/scaleup.yml
```

Figure 41 shows the successful output of scaleup.yml

Figure 54 Output of scaleup.yml



12. Verify the installation. ssh to one of the master nodes and run the following command.

```
 [root@master-0 ~]# oc get nodes
NAME                      STATUS    ROLES     AGE       VERSION
app-0.ocp3.cisco.com      Ready     compute   27d       v1.9.1+a0ce1bc657
app-1.ocp3.cisco.com      Ready     compute   27d       v1.9.1+a0ce1bc657
app-2.ocp3.cisco.com      Ready     compute   27d       v1.9.1+a0ce1bc657
app-3.ocp3.cisco.com      Ready     compute   4m        v1.9.1+a0ce1bc657
infra-0.ocp3.cisco.com    Ready     <none>    27d       v1.9.1+a0ce1bc657
infra-1.ocp3.cisco.com    Ready     <none>    27d       v1.9.1+a0ce1bc657
infra-2.ocp3.cisco.com    Ready     <none>    27d       v1.9.1+a0ce1bc657
master-0.ocp3.cisco.com   Ready     master    27d       v1.9.1+a0ce1bc657
master-1.ocp3.cisco.com   Ready     master    27d       v1.9.1+a0ce1bc657
master-2.ocp3.cisco.com   Ready     master    27d       v1.9.1+a0ce1bc657
storage-01.ocp3.cisco.com Ready     compute   27d       v1.9.1+a0ce1bc657
storage-02.ocp3.cisco.com Ready     compute   27d       v1.9.1+a0ce1bc657
storage-03.ocp3.cisco.com Ready     compute   27d       v1.9.1+a0ce1bc657
[root@master-0 ~]#
```

13. Finally, move any hosts you had defined in the `[new_<host_type>]` section into their appropriate section (but leave the `[new_<host_type>]` section definition itself in place) so that subsequent runs using this inventory file are aware of the nodes but do not handle them as new nodes. For example, when adding new nodes

Add Metrics to the Installation

1. After the installation has been completed, metrics can be added via the following process:

```
$ cat /etc/ansible/hosts
...omitted...
# metrics
openshift_metrics_install_metrics=true
openshift_metrics_hawkular_nodeselector={"role":"infra"}
openshift_metrics_cassandra_nodeselector={"role":"infra"}
openshift_metrics_heapster_nodeselector={"role":"infra"}
openshift_metrics_cassanda_pvc_storage_class_name="glusterfs-registryblock"
openshift_metrics_cassandra_pvc_size=25Gi
openshift_metrics_storage_kind=dynamic
...omitted...
```

2. Run the following playbook:

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

Add Logging to the Installation

To add logging, complete the following steps:

1. Configure /etc/ansible/hosts file with the following:

```
$ cat /etc/ansible/hosts
...omitted...
# logging
openshift_logging_install_logging=true
openshift_logging_es_cluster_size=3
openshift_logging_es_nodeselector={"role":"infra"}
openshift_logging_kibana_nodeselector={"role":"infra"}
openshift_logging_curator_nodeselector={"role":"infra"}
openshift_logging_es_pvc_storage_class_name="glusterfs-registry-block"
openshift_logging_es_pvc_size=10Gi
openshift_logging_storage_kind=dynamic
```

2. Run the following playbook to add logging:

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

Resources

Cisco UCS Infrastructure for Red Hat OpenShift Container Platform Deployment Guide: https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/ucs_openshift.html

Deploying and Managing OpenShift 3.9 on VMware vSphere: https://access.redhat.com/documentation/en-us/reference_architectures/2018/html-single/deploying_and_managing_openshift_3.9_on_vmware_vsphere/index

FlexPod Datacenter with VMware vSphere 6.5 Design Guide: https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/flexpod_esxi65design.html

OpenShift: https://www.openshift.com/

Red Hat OpenShift Container Platform 3.9 Architecture: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/architecture/

Day Two Operations: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/day_two_operations_guide/

CLI Reference: https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html-single/cli_reference/

Red Hat OpenShift: https://www.redhat.com/en/technologies/cloud-computing/openshift

Conclusion

Solution involving Cisco UCS Infrastructure and Red Hat OpenShift Container Platform with container-native storage has been created to deliver a production-ready foundation that simplifies the deployment process, shares the latest best practices, and provides a stable, highly available environment to run your containerized production applications. This reference architecture covers the process of provisioning and deploying a highly available Red Hat OpenShift Container Platform cluster on a VMware vSphere private cloud environment with both the registry and the application pods backed by container-native storage solution from Red Hat.

For enterprise IT, this solution provides a quick and easy journey to DevOps and CI/CD model for application development to address immediate business needs and reducing time to market. Enterprises can accelerate on the path to an enterprise-grade Kubernetes solution with Red Hat OpenShift Container Platform running on Cisco UCS infrastructure.

About the Authors

Muhammad Afzal, Engineering Architect, Cisco UCS Solutions Engineering, Cisco Systems, Inc.

Muhammad Afzal is an Engineering Architect and Technical Marketing Engineer at Cisco Systems in Cisco UCS Product Management and Datacenter Solutions Engineering. He is currently responsible for designing, developing, and producing validated converged architectures while working collaboratively with product partners. Previously, Afzal had been a lead architect for various cloud and data center solutions in Solution Development Unit at Cisco. Prior to this, Afzal has been a Solutions Architect in Cisco's Advanced Services group, where he worked closely with Cisco's large enterprise and service provider customers delivering data center and cloud solutions. Afzal holds an MBA in Finance and a BS in Computer Engineering.

Acknowledgements

- Vishwanath Jakka, Cisco Systems, Inc.
- Babu Mahadevan, Cisco Systems, Inc.
- Antonios Dakopoulos, Red Hat
- Chris Morgan, Red Hat