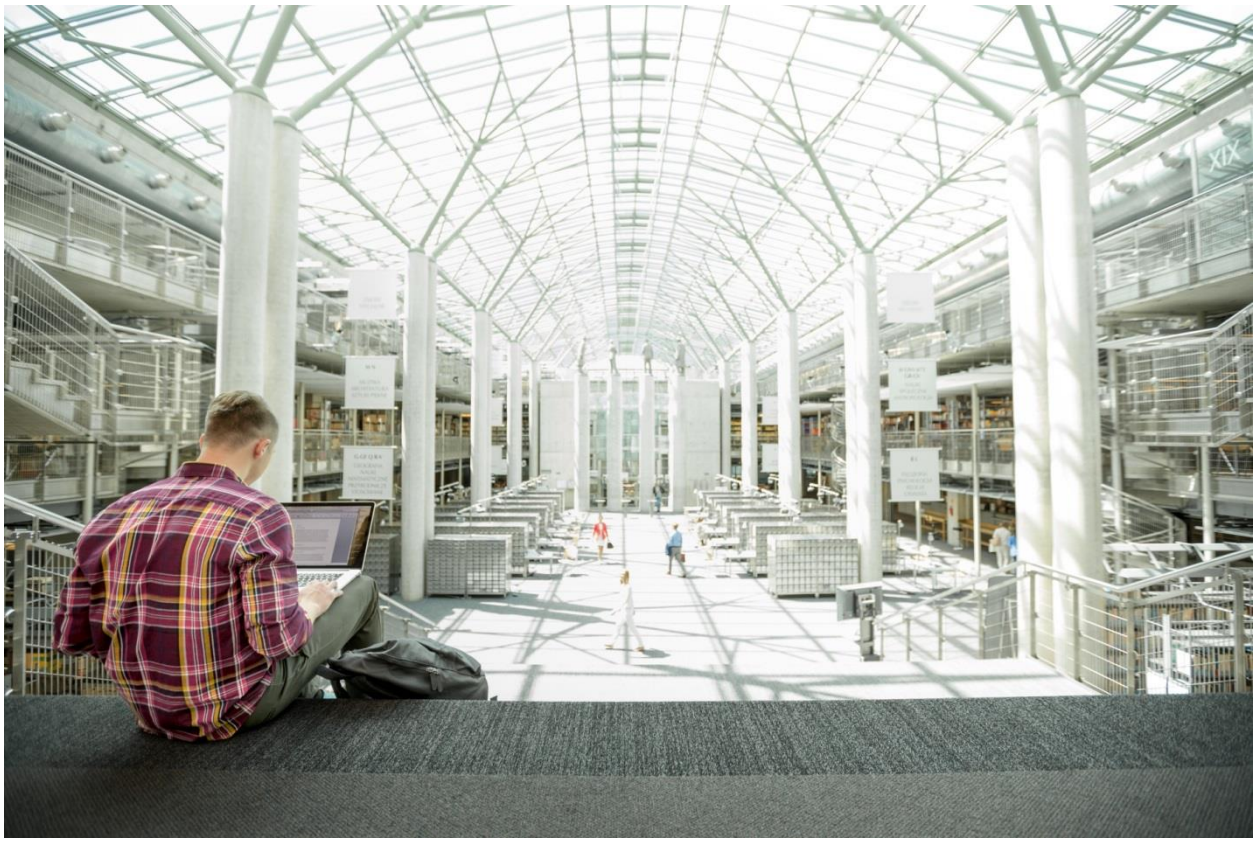


Cisco UCS Infrastructure with Docker Datacenter for Container Management

Design and Deployment Guide for Cisco UCS Infrastructure with Docker Datacenter for Container Management

Last Updated: October 2, 2017

CISCO VALIDATED DESIGN

About the Cisco Validated Design (CVD) Program

The CVD program consists of systems and solutions designed, tested, and documented to facilitate faster, more reliable, and more predictable customer deployments. For more information visit

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

Table of Contents

Executive Summary

Cisco Validated Design program consists of systems and solutions that are designed, tested, and documented to enable end-to-end customer deployments. These designs incorporate a wide range of technologies and products into solution portfolios that have been developed to address the business needs of our customers.

Cisco Unified Computing System™ (Cisco UCS®) servers adapt to meet rapidly changing business requirements, 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).

Docker is an efficient platform for developers and IT operations to use to build, ship, and run distributed applications anywhere. With Microservices architecture shaping the next generation of IT, enterprises with large investments in monolithic applications are finding ways to adopt Docker as a strategy for modernizing their application architectures and keeping the organization competitive and cost effective. Containerization provides the agility, control, and portability that developers and IT operations require to build and deploy applications across any infrastructure. The Docker platform allows distributed applications to be easily composed into a lightweight application container that can change dynamically yet non-disruptively. This capability makes the applications portable across development, test, and production environments running on physical or virtual machines locally, in data centers, and across the networks of different cloud service providers. Docker Datacenter provides native container management tools, including the Docker Trusted Registry (DTR) and Universal Control Plane (UCP) components. It can be deployed on an organization's premises or in a virtual private cloud (VPC) and connected to existing infrastructure and systems such as storage and Lightweight Directory Access Protocol (LDAP) or Microsoft Active Directory (AD) services (LDAP/AD services).

Cisco and Docker have joined hands to offer Container Management Solution on Cisco UCS Infrastructure with Docker Datacenter. This enables enterprises to modernize traditional applications and build Microservices architecture using the Docker platform and tools on Cisco's proven Cisco UCS Integrated Infrastructure. The combination of Docker container technology and Cisco UCS server hardware enables a highly scalable, resilient, and elastic application deployment environment with the simplicity of the on-premise cloud like experience.

Business Challenges

Technological revolution has created new opportunities and challenges for businesses in this digital world. Many startups or smaller competitors are using disruptive innovations such as Microservices architecture to quickly develop and deploy applications and services to rapidly adopt changing markets and meet customer needs. These innovations also provide a path to modernize traditional business critical applications providing agility, flexibility and portability to reduce operational and management costs while improving performance and security. In order to keep up with new technologies or stay one step ahead, enterprises will have to overcome key challenges to accelerate product development, add value and compete better at lower cost.

Key Challenges:

- Portability: Applications have dependencies around OS versions, libraries, Java versions, etc. Any changes to these dependencies can break portability which means applications developed on a specific environment may behave differently on a production environment hosted on-premise or cloud. Changing code and rebuilding/testing code leads to delay in product or service offering and loss of market share.
- Agility: Enterprises have many legacy applications, tools and complex development process slowing innovation significantly as product release cycle takes days to weeks due to complex flow from development to production.
- Resource Utilization: Engineering and hardware resources in large enterprises are not used efficiently due to various organizations operating in silos. These silos that worked in the past are causing a huge overhead in development/release cycle and resource under-utilization as technology changes rapidly.
- Policy Management: Large enterprises have redundant processes and policies in place, and are reluctant to adopt new technologies due to fear of breaking compliance causing delays in new development/release cycle.

Our Solution

This Cisco Validated Design focuses on Cisco UCS Integrated Infrastructure and Docker Datacenter that enables enterprises to rapidly deploy and manage production ready Container as a Service environment on a highly available and secure platform. Additionally, it helps automate DevOps process with an integrated security from development to production, accelerating product release cycle and reducing high OS or VM licensing costs and better resource utilization. The solution is well tested for high-availability, performance and scalability, optimizing time and resources to bring the system and solution online quickly. Cisco has partnered with Docker to deploy Docker Datacenter (which includes Docker Universal Control Plane, Docker Trusted Registry and Commercially Supported- Docker CS Engine) on Cisco UCS platform.

This solution covers production ready installation, provisioning, configuring and deploying application containers using Docker Datacenter on Cisco UCS B-Series and C-Series server platforms in two separate topologies for production and dev/test use cases. Docker Datacenter's Universal Control Plane provide clustering of Docker Engine Nodes and clustering services, automates cluster and application container life cycle management and integrates with Docker Trusted Registry services for application container image services. Cisco UCS infrastructure provides the converged platform for the compute, network, storage and entire hardware lifecycle management through a single management control plane. The solution demonstrates:

- Quick and easy installation of Cisco UCS Integrated Infrastructure, Docker Datacenter and Application Containers
- Application Container Management through Docker Datacenter on compute nodes irrespective of form factors by utilizing Cisco UCS Manager capabilities
- Create and configure network and storage across complete infrastructure for Application Containers
- High-Availability test inducing node and container engine failures
- Scalability apropos of networks, subnets, storage access, containers, and compute/infra nodes

- Performance with regard to reducing the bring-up time of container seen with DTR integration in the stack

The combination of Cisco UCS and Docker Datacenter allows organizations to build and deploy containerized applications on an open, highly available and scalable platform leveraging existing hardware investments to provide an end-end secure platform to meet SLAs.

Implementation Overview

Docker Datacenter solution is integrated and validated on Cisco UCS Integrated Infrastructure. This solution is implemented on Cisco UCS B- and C-Series servers and Cisco Nexus platforms. The architecture covers high level install/configuration, provisioning process and the solution testing required for CVD. Cisco UCS blade and rack servers are provisioned through Service Profiles and OS installation is done manually. OS configuration and Docker Datacenter installation is automated through built-in Docker tools and Ansible. The end-to-end stack is tested for correctness (recommended software stack), performance, and scalability, high-availability and security policies. The containers are deployed and managed by Docker UCP. The deployment guide provides step by step instructions on setting up the complete stack and solution validation test results.

Highlights

- Industry-Leading Converged Computing Infrastructure: Cisco UCS blade and rack servers enable customers to rapidly and efficiently deploy and manage compute, network and storage functions for containers. They enable customers to reduce IT costs through consolidation, manage more containers per computing node and make better use of infrastructure to provide an application container environment in real time.
- Industry-Leading Container Application Infrastructure: Docker Datacenter brings container orchestration, management and security to the enterprise. It enables developers and system administrators to build, ship and run distributed applications anywhere.
- Combined Use cases for the enterprise who want to leverage the solution:
  - Micro-service and cloud native application architecture: Enables stateless application container deployment for specific business use needs
  - Continuous integration and continuous deployment(CI/CD): Enable developers to develop and test applications more quickly and within relevant environment
  - DevOps: Break down barriers between development and operations teams to improve delivery of applications for the business

  This platform allows you to quickly adopt CI/CD and DevOps use cases by adding components like Jenkins and integrating it with Docker Datacenter. It also provides an infrastructure platform for converting applications into Microservices and cloud native architectures and deploying them through stateless application containers.
- Big Data: Empower organizations to use big data analytics using small foot print applications at a very large scale numbers
  - Infrastructure optimization: Decrease infrastructure costs while increasing efficiency. The lightweight nature of containers and the absence of hypervisors and guest operating systems enables enterprises to optimize resource and eliminate licensing costs

Solution Benefits

The benefits of Cisco UCS with Docker Datacenter include the following:

- Cisco UCS
  - Simplicity: Reduced datacenter complexities through Cisco UCS converged infrastructure with a single management control plane for hardware lifecycle management
  - Rapid Deployment: Easily deploy and scale the solution
  - High Availability: Superior scalability and high-availability
  - Flexibility: Compute form factor agnostic
  - Faster ROI: Better response with optimal ROI
  - Resource Utilization: Optimized hardware footprint for production and dev/test deployments
- Docker Datacenter
  - Agility: Gain the freedom to define environments and create and deploy applications quickly and easily, providing flexibility of IT operations that respond quickly to change
  - Control: Enable developers to own the code from the infrastructure to the application and quickly move from the build to the production environment. IT operations manageability features enable organizations to standardize, secure, and scale the operating environment
  - Portability: Docker Containers are self-contained and independent units that are portable between private infrastructure and public cloud environments without complexity or disruption

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 (UCS) and Cisco Nexus Switches as well as high level understanding of Docker Container components. External references are provided where applicable and it is recommended that the reader be familiar with these documents.

Readers are also expected to be familiar with the infrastructure, network and security policies of the customer installation.

Purpose of the Document

This document highlights the benefits of using Cisco UCS infrastructure with Docker Datacenter to efficiently deploy, scale, and manage a production-ready application container environment for enterprise customers. The goal of this document is to demonstrate the value that Cisco UCS brings to the datacenter, such as single-point hardware lifecycle management and highly available converged compute and network infrastructure for application container deployments using Docker Datacenter.

Solution Overview

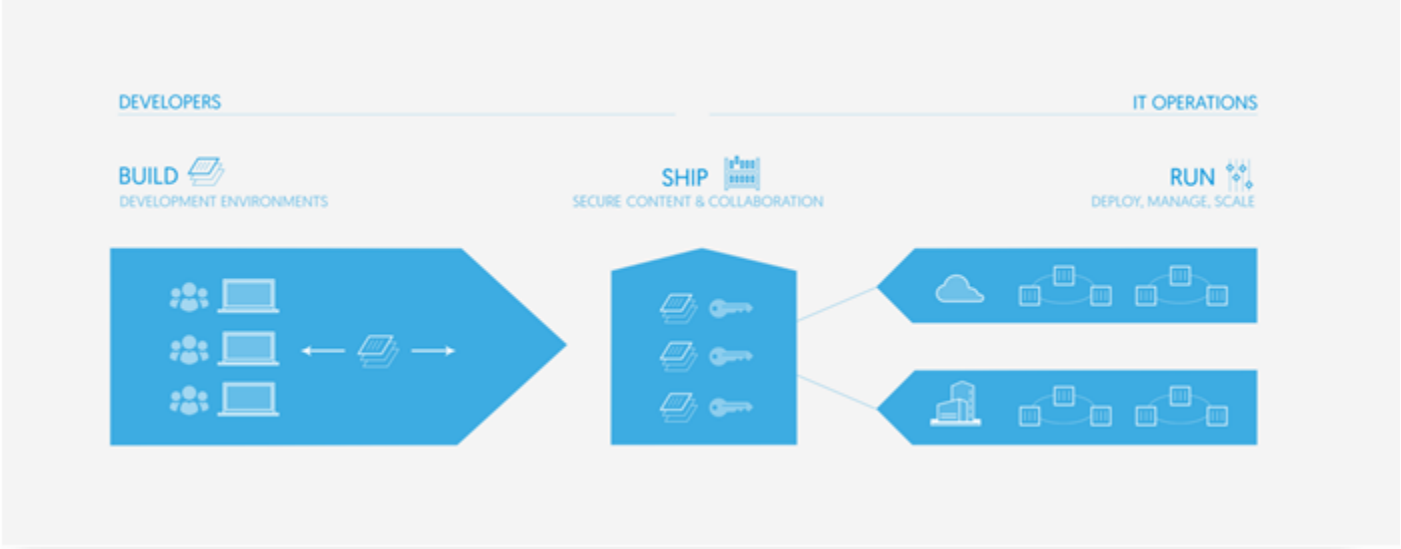
Cisco UCS Integrated Infrastructure solutions speed up IT operations today and create the modern technology foundation you need for initiatives like private cloud, big data, and desktop virtualization. With Cisco UCS Manager and Cisco Single Connect Technology, hardware is automatically configured by application-centric policies–ushering in a new era of speed, consistency, and simplicity for datacenter operations. UCS brings the flexibility of virtualized systems to the physical world in a way no other server architecture can, lowering costs and improving your ROI.

Leveraging the centralized management of Cisco UCS Manager, this solution provides unified, embedded, policy-driven management to programmatically control server, network, and storage resources you can efficiently manage the scale-up/ -out infrastructure. Furthermore, Cisco Nexus - unified Fabric is a holistic network architecture comprising switching, security, and services that are designed for physical, virtual, and cloud environments. It uniquely integrates with servers, storage, and orchestration platforms for more efficient operations and greater scalability.

Cisco has partnered with Docker to provide Container Management solution to accelerate the IT transformation by enabling easy and faster deployments, greater flexibility of choice, business agility, efficiency, lower risk.

Docker has become the industry standard for developers and IT operations to build, ship and run distributed applications in baremetal, virtualized and cloud environments. As organizations adopt public, private or Hybrid cloud, Docker makes it easy to move applications between on premise and cloud environments. Docker can significantly improve hardware resource utilization, accelerate application lifecycle and reduce overall cost by automating IT processes, and deploying dockerized applications on-premise or in cloud environment.

Figure 1 High-Level Docker Infrastructure Diagram



Docker Datacenter delivers an integrated platform for developers and IT operations to collaborate in the enterprise software supply chain. Bringing security, policy and controls to the application lifecycle without sacrificing any agility or application portability. Docker Datacenter integrates to enterprise business – from on-premises and VPC deployment models, open APIs and interfaces, to flexibility for supporting a wide variety of workflows.

Solution Components

The solution offers redundant architecture from a compute, network, and storage perspective. The solution consists of the following key components:

- Cisco Unified Computing System (UCS)
- Cisco UCS Manager
- Cisco UCS 6248UP Fabric Interconnects
- Cisco 2208XP IO Module or Cisco UCS Fabric Extenders
- Cisco B200 M4 Servers
- Cisco C220 M4S Servers
- Cisco VIC 1340
- Cisco VIC 1227
- Cisco Nexus 9372PX Switches
- Docker Datacenter (DDC)
- Docker CS Engine
- Docker Universal Control Plane (UCP)
- Docker Trusted Repository (DTR)
- Red Hat Enterprise Linux 7.2

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 26xx v4 Series Processors.
- Network - The system is integrated onto a low-latency, lossless, 10-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 (UCS) Manager provides unified, embedded management for all software and hardware components in the 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 6200 Series Fabric Interconnects using a clustered, active-standby configuration for high-availability.

UCS Manager offers unified embedded management interface that integrates server, network, and storage. 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 Cisco UCS Manager Information, refer to: 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 6200 Series Fabric Interconnects support the system's 80-Gbps 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 6248UP is a 1-RU Fabric Interconnect that features up to 48 universal ports that can support 80 Gigabit Ethernet, Fiber Channel over Ethernet, or native Fiber Channel connectivity.

For more information, visit the following link: http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-6200-series-fabric-interconnects/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 2204XP or 2208XP Fabric Extenders. A passive mid-plane provides up to 40 Gbps of I/O bandwidth per server slot and up to 80 Gbps of I/O bandwidth for two slots. The chassis is capable of supporting future 80 Gigabit Ethernet standards.

For more information, please refer to the following link: http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-5100-series-blade-server-chassis/index.html

Cisco UCS B200M4 Blade Server

The enterprise-class Cisco UCS B200 M4 blade server extends the capabilities of Cisco's Unified Computing System portfolio in a half-width blade form factor. The Cisco UCS B200 M4 uses the power of the latest Intel® Xeon® E5-2600 v3 Series processor family CPUs with up to 768 GB of RAM (using 32 GB DIMMs), two solid-state drives (SSDs) or hard disk drives (HDDs), and up to 80 Gbps throughput connectivity. The UCS B200 M4 Blade Server mounts in a Cisco UCS 5100 Series blade server chassis or UCS Mini blade server chassis. It has 24 total slots for registered ECC DIMMs (RDIMMs) or load-reduced DIMMs (LR DIMMs) for up to 768 GB total memory capacity (B200 M4 configured with two CPUs using 32 GB DIMMs). It supports one connector for Cisco's VIC 1340 or 1240 adapter, which provides Ethernet and FCoE.

For more information, see: http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-b200-m4-blade-server/index.html

Cisco UCS C220M4 Rack-Mount Server

The enterprise-class Cisco UCS C220 M4 server extends the capabilities of the Cisco Unified Computing System (UCS) portfolio in a one rack-unit (1RU) form-factor. The Cisco UCS C220 M4 uses the power of the latest Intel® Xeon® E5-2600 v3 and v4 Series processor family CPUs with up to 1536 GB of RAM (using 64 GB DIMMs), 8 Small Form-Factor (SFF) drives or 4 Large Form-Factor (LFF) drives, and up to 80 Gbps throughput connectivity. The Cisco UCS C220 M4 Rack Server can be used standalone, or as integrated part of the Unified Computing System. It has 24 DIMM for up to 1536 GB total memory capacity. It supports one connector for the Cisco VIC 1225, 1227 or 1380 adapters, which provide Ethernet and FCoE.

For more information, see: http://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-c220-m4-rack-server/datasheet-c78-732386.html

Cisco UCS Fabric Extenders

The Cisco UCS 2208XP Fabric Extender has eight 10 Gigabit Ethernet, FCoE-capable, and Enhanced Small Form-Factor Pluggable (SFP+) ports that connect the blade chassis to the fabric interconnect. Each Cisco UCS 2208XP has thirty-two 10 Gigabit Ethernet ports connected through the mid-plane to each half-width slot in the chassis. Typically configured in pairs for redundancy, two fabric extenders provide up to 160 Gbps of I/O to the chassis.

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 40 Gb 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 1227 offers a modular LAN on Motherboard (mLOM) adapter. The mLOM slot, new to Cisco rack servers, can be used to install a Cisco VIC without consuming a PCIe slot. This provides greater I/O expandability.

A Cisco innovation, the Cisco UCS Virtual Interface Card (VIC) 1227 is a dual-port, Enhanced Small Form-Factor Pluggable (SFP+), 10 Gigabit Ethernet and Fibre Channel over Ethernet (FCoE)-capable, PCI Express (PCIe) modular LAN on motherboard (mLOM) adapter. It is designed exclusively for the M4 generation of Cisco UCS C-Series Rack Servers and the Cisco UCS S-Series dense storage servers.

For more information, see:

http://www.cisco.com/c/en/us/products/interfaces-modules/ucs-virtual-interface-card-1227/index.html

Cisco UCS Differentiators

Cisco's Unified Compute System is revolutionizing the way servers are managed in data-center. Following are the unique differentiators of UCS and UCS Manager:

1. Embedded Management –In UCS, the servers are managed by the embedded firmware in the Fabric Interconnects, eliminating need for any external physical or virtual devices to manage the servers.
2. Unified Fabric –In UCS, from blade server chassis or rack servers to FI, there is a single Ethernet cable used for LAN, SAN and management traffic. This converged I/O results in reduced cables, SFPs and adapters – reducing capital and operational expenses of overall solution.
3. Auto Discovery –By simply inserting the blade server in the chassis or connecting rack server to the fabric interconnect, discovery and inventory of compute resource occurs automatically without any management intervention. The combination of unified fabric and auto-discovery enables the wire-once architecture of UCS, where compute capability of UCS can be extended easily while keeping the existing external connectivity to LAN, SAN and management networks.
4. Policy Based Resource Classification –Once a compute resource is discovered by UCS Manager, it can be automatically classified to a given resource pool based on policies defined. This capability is useful in multi-tenant cloud computing. This CVD showcases the policy based resource classification of UCS Manager.
5. Combined Rack and Blade Server Management –UCS Manager can manage B-Series blade servers and C-Series rack server under the same UCS domain. This feature, along with stateless computing makes compute resources truly hardware form factor agnostic.

6. Model based Management Architecture –UCS Manager Architecture and management database is model based and data driven. An open XML API is provided to operate on the management model. This enables easy and scalable integration of UCS Manager with other management systems.
7. Policies, Pools, Templates –The management approach in UCS Manager is based on defining policies, pools and templates, instead of cluttered configuration, which enables a simple, loosely coupled, data driven approach in managing compute, network and storage resources.
8. Loose Referential Integrity –In UCS Manager, a service profile, port profile or policies can refer to other policies or logical resources with loose referential integrity. A referred policy cannot exist at the time of authoring the referring policy or a referred policy can be deleted even though other policies are referring to it. This provides different subject matter experts to work independently from each-other. This provides great flexibility where different experts from different domains, such as network, storage, security, server and virtualization work together to accomplish a complex task.
9. Policy Resolution –In UCS Manager, a tree structure of organizational unit hierarchy can be created that mimics the real life tenants and/or organization relationships. Various policies, pools and templates can be defined at different levels of organization hierarchy. A policy referring to another policy by name is resolved in the organization hierarchy with closest policy match. If no policy with specific name is found in the hierarchy of the root organization, then special policy named "default" is searched. This policy resolution practice enables automation friendly management APIs and provides great flexibility to owners of different organizations.
10. Service Profiles and Stateless Computing –a service profile is a logical representation of a server, carrying its various identities and policies. This logical server can be assigned to any physical compute resource as far as it meets the resource requirements. Stateless computing enables procurement of a server within minutes, which used to take days in legacy server management systems.
11. Built-in Multi-Tenancy Support –The combination of policies, pools and templates, loose referential integrity, policy resolution in organization hierarchy and a service profiles based approach to compute resources makes UCS Manager inherently friendly to multi-tenant environment typically observed in private and public clouds.
12. Extended Memory – the enterprise-class Cisco UCS B200 M4 blade server extends the capabilities of Cisco's Unified Computing System portfolio in a half-width blade form factor. The Cisco UCS B200 M4 harnesses the power of the latest Intel® Xeon® E5-2600 v4 Series processor family CPUs with up to 1536 GB of RAM (using 64 GB DIMMs) – allowing huge VM to physical server ratio required in many deployments, or allowing large memory operations required by certain architectures like Big-Data.
13. Virtualization Aware Network –VM-FEX technology makes the access network layer aware about host virtualization. This prevents domain pollution of compute and network domains with virtualization when virtual network is managed by port-profiles defined by the network administrators' team. VM-FEX also off-loads hypervisor CPU by performing switching in the hardware, thus allowing hypervisor CPU to do more virtualization related tasks. VM-FEX technology is well integrated with VMware vCenter, Linux KVM and Hyper-V SR-IOV to simplify cloud management.
14. Simplified QoS –Even though Fiber Channel and Ethernet are converged in UCS fabric, built-in support for QoS and lossless Ethernet makes it seamless. Network Quality of Service (QoS) is simplified in UCS Manager by representing all system classes in one GUI panel.

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.

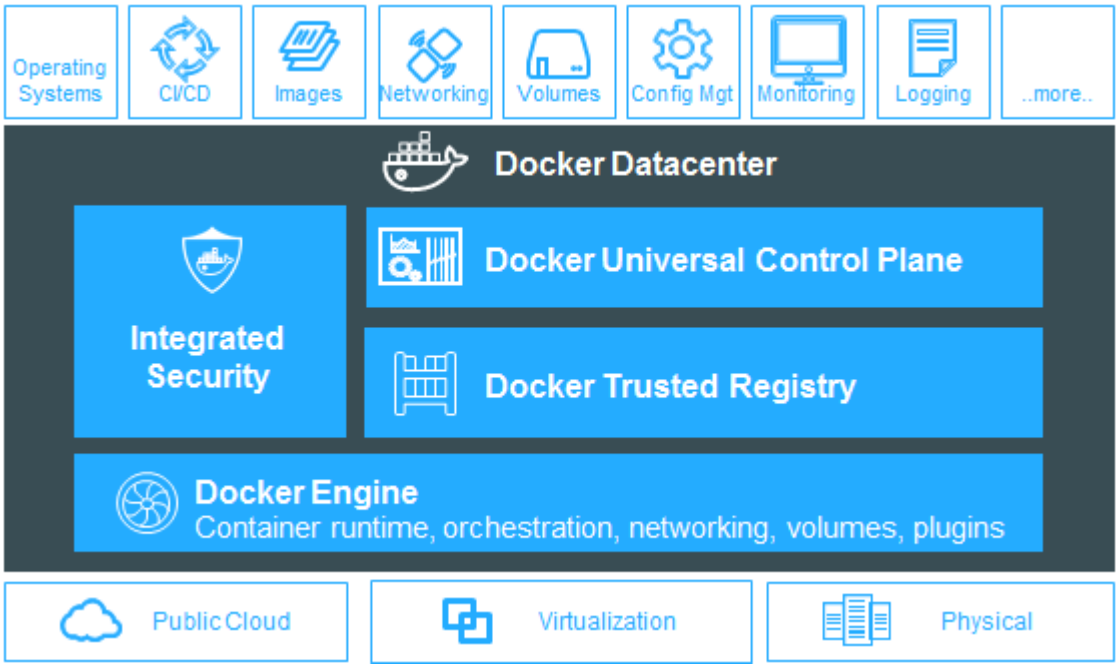
Docker Datacenter

Docker - a containerization platform developed to simplify and standardize deployment in various environments. It is largely instrumental in spurring the adoption of this style of service design and management. Docker containers encapsulate all application components, such as dependencies and services. When all dependencies are encapsulated, applications become portable and can be dependably moved between development, test, and production environments. Docker makes container creation and management simple and integrates with many open source projects. Docker Datacenter comprises an enterprise container orchestration, application management and enterprise-grade security.

Docker Datacenter includes leading Docker open source projects, commercial software, integrations with validated and supported configurations:

- Commercially supported Docker Engine for a robust container runtime
- Universal Control Plane (UCP) with embedded Swarm scheduler for integrated management and orchestration of the Docker environment
- Trusted Registry (DTR) for Docker image management, security, and collaboration
- Security must be a multi-layered approach; content Trust provides the ability to sign images with digital keys and then verify the signature of those images

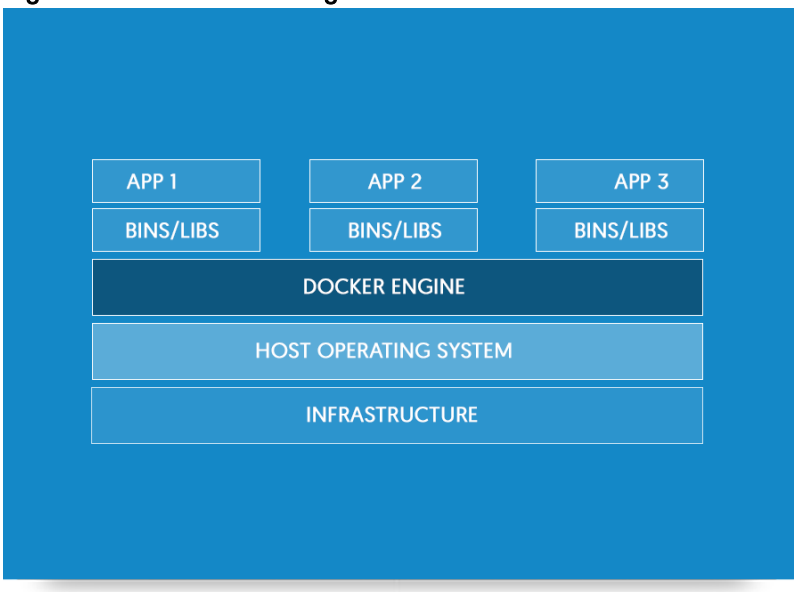
Figure 2 Docker Datacenter Platform



Docker CS Engine

Docker engine is the building block for the modern application platform. The Docker CS Engine is commercially supported version of Docker Engine for the enterprise. It's a lightweight container runtime and robust tooling that runs and build containers. Docker allows us to package the application code and dependencies together in an isolated container that shares the OS kernel on the host system. The in-host daemon communicates with Docker Client to execute commands to build, ship and run containers.

Figure 3 Docker CS Engine

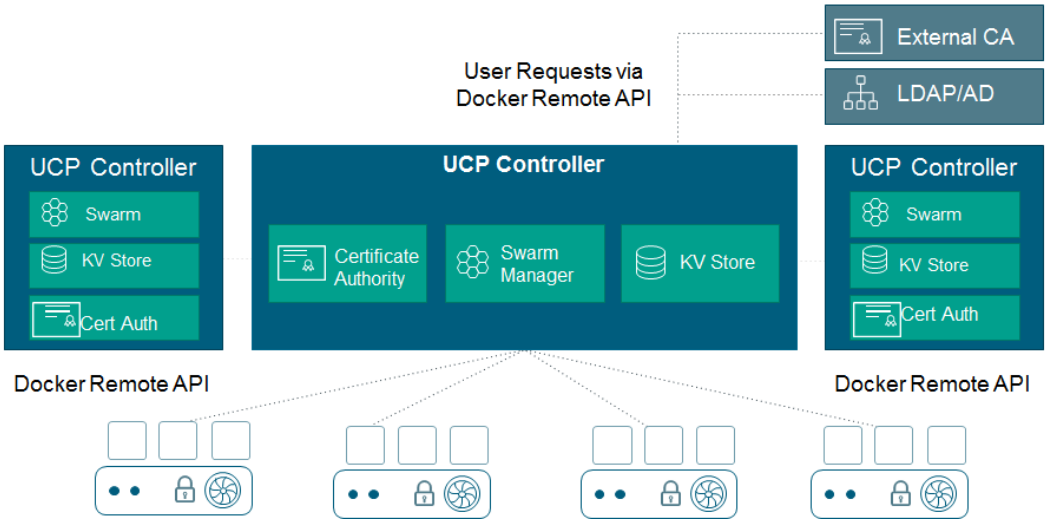


Docker UCP

Docker Universal Control Plane (Docker UCP) is an enterprise on premise solution that includes user management, resource management, clustering and orchestration that integrates with the existing enterprise LDAP/AD for High-Availability, security and compliance. Docker UCP enables IT operation teams to deploy and manage the containerized applications in production.

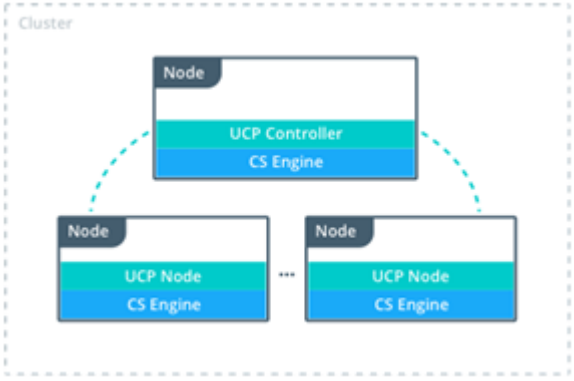
The figure below shows the Docker UCP architecture and illustrates the built-in HA for UCP.

Figure 4 Docker UCP Architecture



UCP is a containerized application, which provides enterprise-grade cluster management solution. It gets installed on Docker CS Engine on all the nodes which are designated as controller nodes. After installing first node with Docker Engine, UCP application is installed and the other nodes are joined to the first node to form a Docker Datacenter cluster.

Figure 5 Docker Datacenter Cluster



A Docker UCP cluster has two types of nodes:

- UCP Controller Node
- UCP Node

UCP Controller Node

Docker UCP controller node manages the cluster and provides persistent cluster configurations. Within controller nodes there are two categories of nodes – Master and Replicas. A first node which gets installed

with UCP is treated as a Master Controller node. And controller nodes joining the master controller node are termed as Replica nodes. Controller nodes can take up application container workload as well. This is a configurable option available at the admin and user level.

Figure 6 Docker UCP Controller

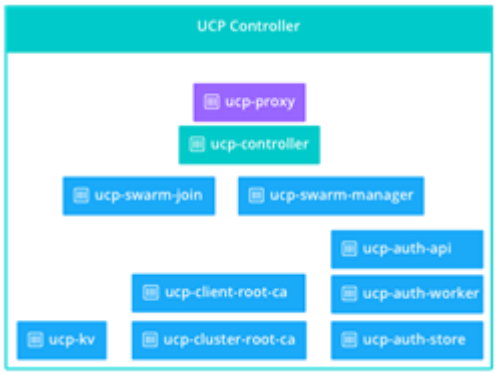


Table 1 Docker UCP Controller node containers and their description

Name	Description
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine.
ucp-controller	The UCP application. It uses the key-value store for persisting configurations.
ucp-swarm-manager	Provides the clustering capabilities. It uses the key-value store for leader election, and keeping track of cluster members.
ucp-swarm-join	Heartbeat to record on the key-value store that this node is alive. If the node goes down, this heartbeat stops, and the node is removed from the cluster.
ucp-auth-api	The centralized API for identity and authentication used by UCP and DTR.
ucp-auth-worker	Performs scheduled LDAP synchronizations and cleans data on the ucp-auth-store.
ucp-auth-store	Stores authentication configurations, and data for users, organizations, and teams.
ucp-kv	Used to store the UCP configurations. Do not use it in your applications, since it's for the internal use only.
ucp-cluster-root-ca	A certificate authority to sign the certificates used when joining new nodes, and on administrator client bundles.
ucp-client-root-ca	A certificate authority to sign user bundles. Only used when UCP is installed without an external root CA.

Table 2 Following are the named volumes used by Docker UCP for persistent data

Node	Volume name	Location on host [var/lib/docker/volumes/]	Description

all	ucp-client-root-ca	ucp-client-root-ca/_data	The certificate and key for the UCP root CA. Do not create this volume if you are using your own certificates.
all	ucp-cluster-root-ca	ucp-cluster-root-ca/_data	The certificate and key for the Swarm root CA.
all	ucp-controller-client-certs	ucp-controller-client-certs/_data	The UCP Controller Swarm client certificates for the current node.
all	ucp-controller-server-certs	ucp-controller-server-certs/_data	The controller certificates for the UCP controllers web server.
all	ucp-kv	ucp-kv/_data	Key value store persistence.
all	ucp-kv-certs	ucp-kv-certs/_data	The Swarm KV client certificates for the current node (repeated on every node in the cluster).
all	ucp-node-certs	ucp-node-certs/_data	The Swarm certificates for the current node (repeated on every node in the cluster).

While installing Docker UCP these volumes get created with default volume driver. In our solution we have used Docker device-mapper driver in direct-lvm mode.

Docker Universal Control Plane works in high-availability mode. Adding replicas to first UCP Controller node makes the cluster HA ready. A minimum three-node Controller cluster is needed to tolerate one node failure. Adding replica nodes to the cluster allows user request to get load-balanced across controller master and replica nodes.

Docker UCP does not include load-balancing services. It needs external load-balancer to balance user requests across all the controller nodes.

Docker UCP Node

Docker UCP nodes take the container workload. These are the nodes where containers get deployed. Nodes with Docker CS Engine join the existing UCP cluster. While joining the existing UCP Cluster the following containers get instantiated on the UCP node.

Figure 7 Containers on UCP node while joining the UCP cluster

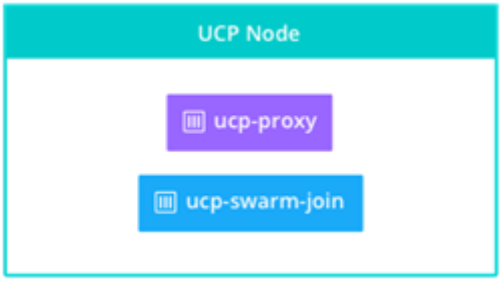


Table 3 Containers formed during UCP Node Join

Name	Description
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine.
ucp-swarm-join	Heartbeat to record on the key-value store that this node is alive. If the node goes down, this heartbeat stops, and the node is dropped from the cluster.

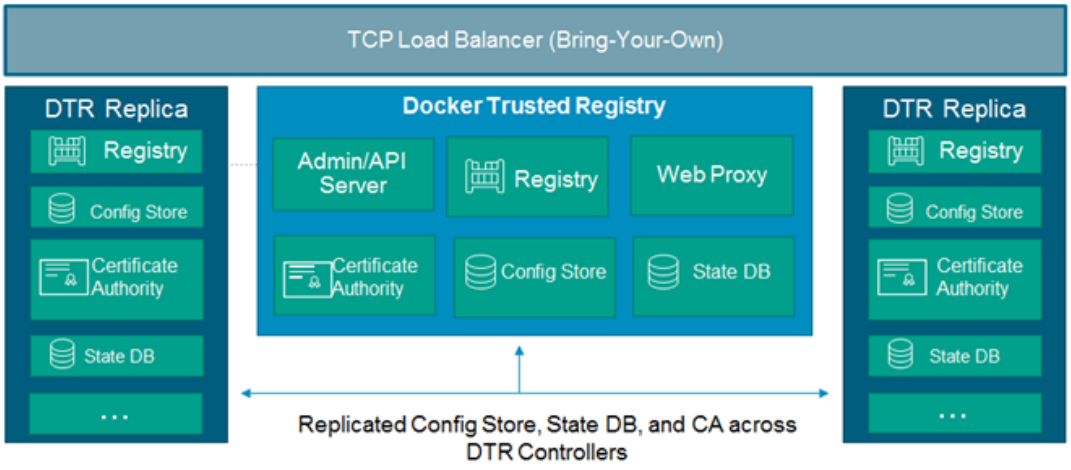
Docker Trusted Registry

Docker Trusted Registry (DTR) is an enterprise-grade image storage solution from Docker. DTR gives enterprises the security and compliance to store and manage their Docker images on-premises or in their virtual private cloud (VPC). It has a built-in authentication mechanism; supports role based access control, and allows integration with LDAP and Active Directory.

DTR is part of the Docker Datacenter Subscription which also includes, Universal Control Plane, and commercially supported Engine. DTR is easy to deploy, configure and integrate with your existing infrastructure and application delivery workflows.

The figure below illustrates the built-in HA for DTR within the control plane. HA ensures continuous availability for Docker Trusted Registry; in the event of the main instance failure, a replica instance takes over non-disruptively.

Figure 8 Docker Trusted Registry Architecture



Docker Trusted Registry (DTR) is a dockerized application that runs a Docker Universal Control Plane cluster.

Figure 9 DTR as Dockerized application in the UCP cluster

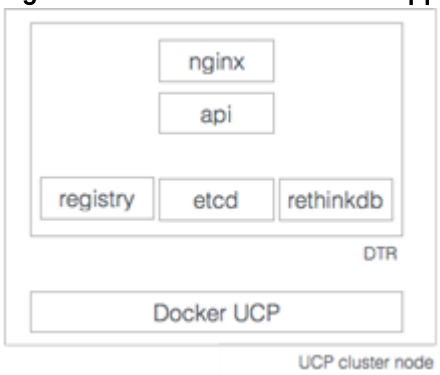


Table 4 With DTR installation the following containers are started

Name	Description
dtr-nginx-<replica_id>	Receives http and https requests and proxies them to other DTR components. By default it listens to ports 80 and 443 of the host.
dtr-api-<replica_id>	Executes the DTR business logic. It serves the DTR web application, and API.
dtr-registry-<replica_id>	Implements the functionality for pulling and pushing Docker images. It also handles how images are stored.
dtr-etcd-<replica_id>	A key-value store for persisting DTR configuration settings. Don't use it in your applications, since it's for internal use only.
dtr-rethinkdb-<replica_id>	A database for persisting repository metadata. Don't use it in your applications, since it's for internal use only.

The communication between all DTR components is secured using TLS. Also, when installing DTR, two certificate authority (CA) are created. These CAs are used to create the certificates used by etcd and rethinkDB when communicating across nodes.

Table 5 Following networking gets created when DTR containers communicate with each other

Name	Type	Description
dtr-br	bridge	Allows containers on the same node to communicate with each other in a secure way.
dtr-ol	overlay	Allows containers running on different nodes to communicate. This network is used in high-availability installations, to allow etcd and rethinkDB containers to replicate their data.

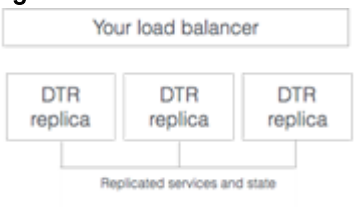
Table 6 DTR uses following named volumes for persistent data

Volume name	Location on host (/var/lib/docker/volumes/)	Description
dtr-ca-<replica_id>	dtr-ca/_data	The volume where the private keys and certificates are stored so that containers can use TLS to communicate.
dtr-etcd-<replica_id>	dtr-etcd/_data	The volume used by etcd to persist DTR configurations.
dtr-registry-<replica_id>	dtr-registry/_data	The volume where images are stored, if DTR is configured to store images on the local filesystem.
dtr-rethink-<replica_id>	dtr-rethink/_data	The volume used by RethinkDB to persist DTR data, like users and repositories.

For shared image repo, data between the DTR cluster 'dtr-registry/_data' can be NFS mounted for high-availability.

For load balancing and high-availability, multiple replicas of DTR are installed and get joined to form a cluster, within Docker Datacenter UCP cluster. When joining new replicas to the cluster, we create new DTR instances that are running the same set of services. Any change to the state of an instance is replicated across all the other instances.

Figure 10 DTR Cluster behind load balancer



Having a DTR cluster with multiple replicas, allows us to:

- Load-balance user requests across the DTR replicas
- Keep the DTR cluster up and running when a replica fails

DTR does not provide a load balancing service. An on-premises or cloud-based load balancer is required to balance requests across multiple DTR replicas. Load balancer is required to be configured:

- On TCP ports 80 and 443
- On a TCP load balancer which does not terminates the HTTPS connections

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

Architectural Overview

This solution consists of two alternate topologies, targeted for production and dev/test deployments. For production environment, we propose an architecture having Cisco UCS B-series servers, with nodes dedicated for Docker Universal Control Plane (UCP) Controller and Docker Trusted Registry (DTR) services within Docker Datacenter. This document also provides a second architecture with a minimal number of servers to implement Docker Datacenter on Cisco UCS using C-Series rack servers. This alternative is a best fit for typical dev/test scenarios for example. Cisco UCS Fabric Interconnects and Cisco Nexus TOR switches are used in both the topologies separately. The Docker Datacenter runs on Red Hat Enterprise Linux Operating System. Cisco UCS servers provide converged and highly available hardware platform centrally managed by Cisco UCS Manager Software residing on Cisco Fabric Interconnects. As an important component of the Docker Datacenter product, Docker Universal Control Plane provides the redundancy and high-availability of the Docker Engine and management control plane. This solution holistically offers container management for diverse application environment to be deployed in DevOps and Production deployments.

Compute nodes for Docker are configured to run on Cisco B- and C-Series servers based on the selected reference architecture. In some cases, compute node resources could be shared between the control plane software stack and the container engine. For production deployments, Docker Universal Control Plane services are spread over three B-Series servers for providing management control plane redundancy and high-availability. DTR services and its replicas are configured to run on three dedicated B-Series servers as part of Docker Datacenter.

In the second architecture Docker UCP and DTR services are co-located on three C-Series rack servers. Docker UCP administrator has a choice to allow distribution of application container work-load to be spread across the Docker UCP controller nodes.

Both Docker UCP and DTR dashboards require an external load balancer to access management interfaces to make them operate in highly available mode. External load balancer provides virtual IP address to front end the UCP and DTR management dashboard separately. External load balancer is not listed as a solution component in this CVD as the load balancer is purely a customer's choice and configuration follows a standard procedure.

To achieve DTR shared storage high availability for image repository among the DTR cluster nodes, Docker Datacenter requires external NFS setup. This solution uses NFS shared volume configuration for the DTR shared storage.

Sizing Considerations

This solution is designed and proposes two separate topologies based on Docker Datacenter (DDC) production as well as dev/test deployment requirements for the Enterprise.

For the production grade DDC deployment Docker recommends infrastructure services to be run on dedicated hosts. There are two infrastructure components, namely – Docker UCP Controller and DTR. Container workloads are to run on dedicated node termed as Docker UCP Nodes. For production requirements, one each of Docker UCP Controller and DTR service node should be running in cluster of baremetal servers. In order to have a minimum of one node failure tolerance, it is recommended to have

three-nodes for running both controller and DTR services separately. Based on this recommendation, Docker UCP Controller and DTR services will run on three-node dedicated clusters. This solution proposes a ten node setup, where six nodes are consumed by the DDC's infrastructure (controller and DTR) services and the remaining four nodes take up the Container workload. Administrators have an option to configure the UCP Controller nodes to take the container workload based on the deployment requirements. With this configurable item, the entire infrastructure is well optimized for running container workloads.

Cisco C-Series servers have ample memory and CPU resources that allow us to run the Docker UCP and DTR service containers on a same three-node cluster. With this design the overall Docker Datacenter deployment gets optimized to a four-node cluster configuration. The fourth node runs as a UCP node to take the container workload. The scheduler configuration settings allows administrators and users to deploy containers on the UCP controller nodes and using the scheduling strategy set to 'spread', we get all the four nodes available for container deployment.

Number of containers run on a node and thus the optimal achievable number from this topology depends on the type of containers and the applications that are run within the containers. This solution is designed and validated to run 300 containers per UCP Node. Adding is a node to the cluster follows a simple procedure with a minimal manual intervention. With a policy based logical server model, achieved through UCS manager, scaling-up the baremetal nodes are just a few clicks away. Service profile templates associated with configured server pool help in automatic deployment of service profiles, as long as there is additional hardware available in the server pool. Node addition workflow is covered in the section Upscale Tests.

The following figure illustrates the physical backend connectivity of the hardware components.

Figure 11 Physical Topology – First Architecture

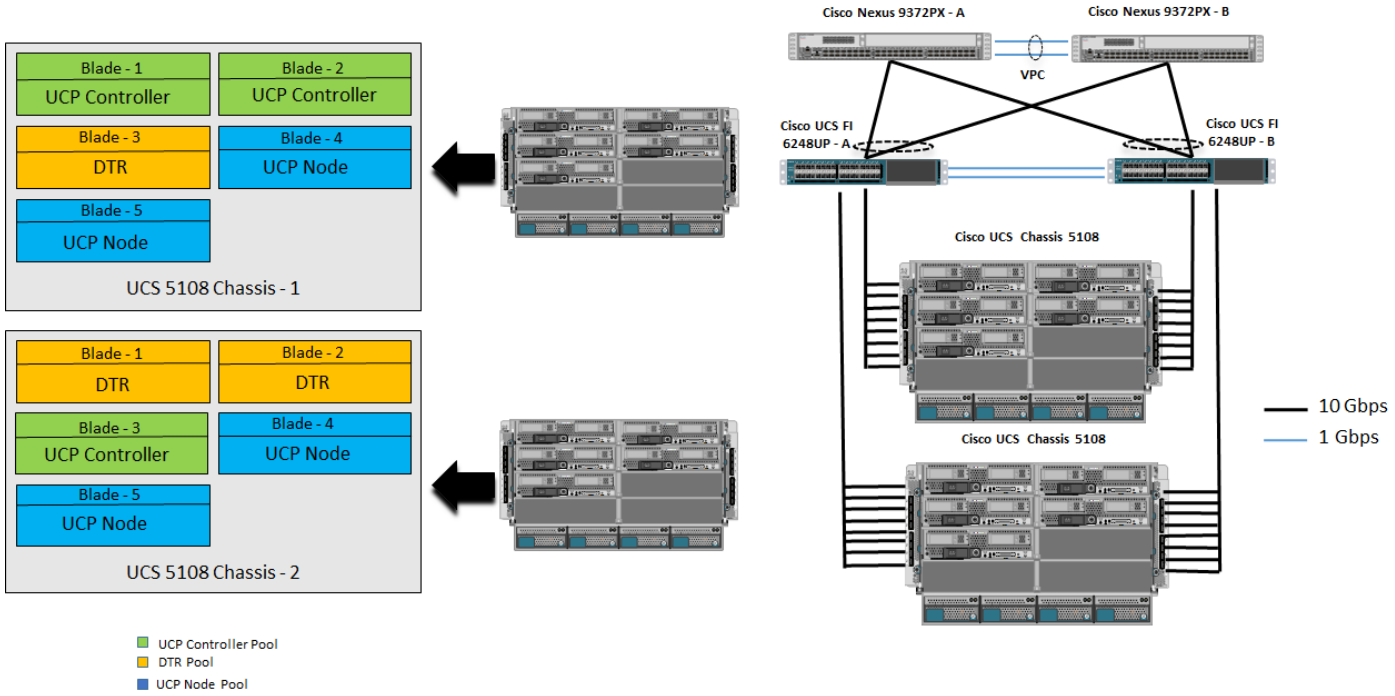


Figure 12 Cabling Diagram – First Architecture

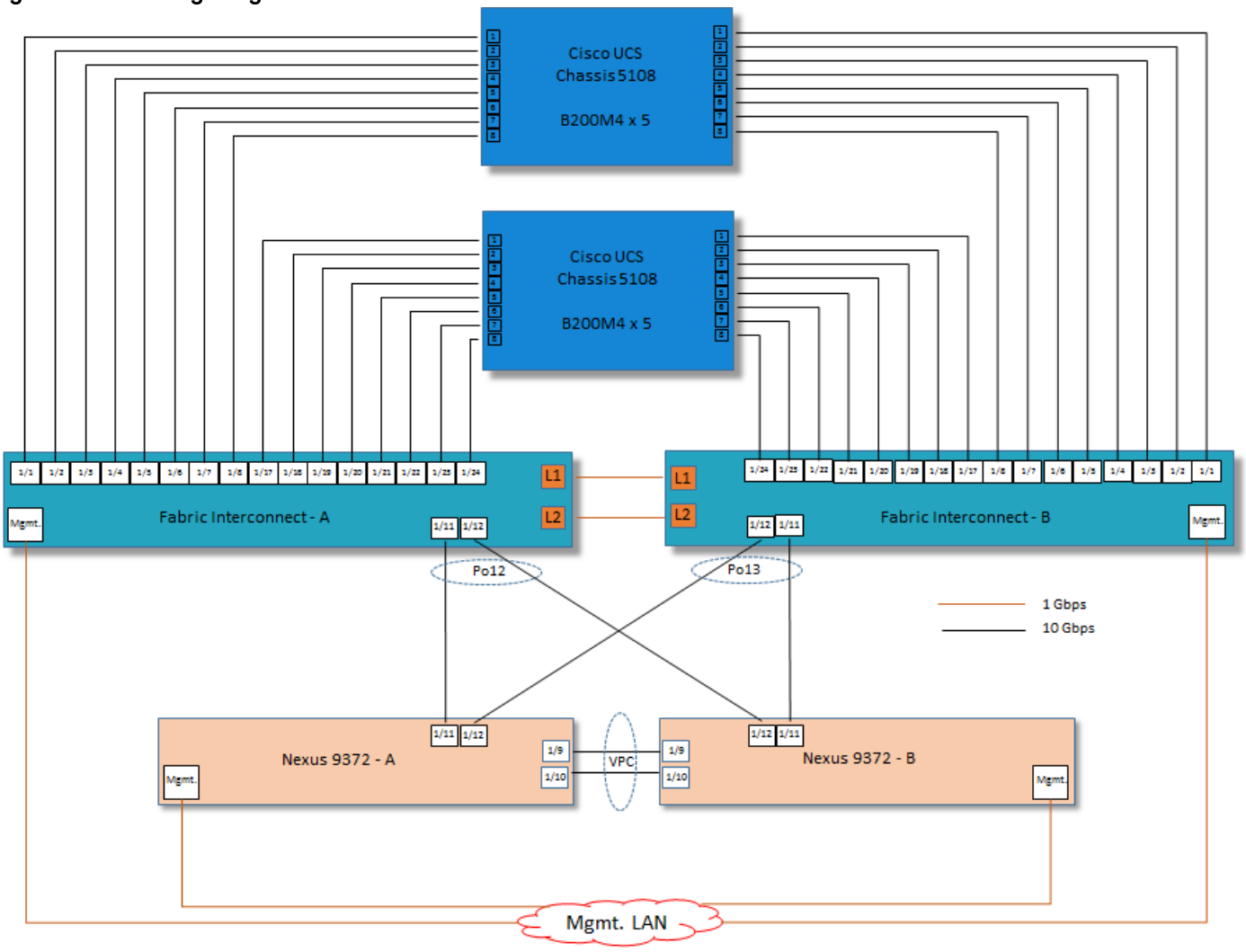
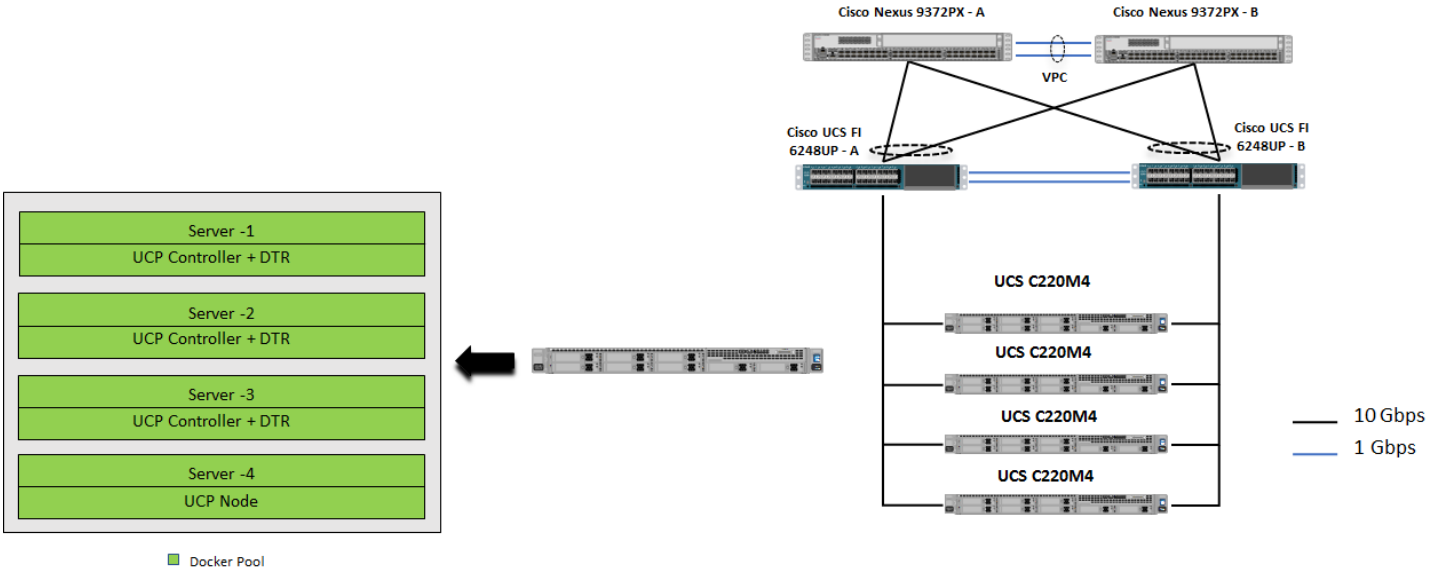


Figure 13 Physical Topology – Second Architecture

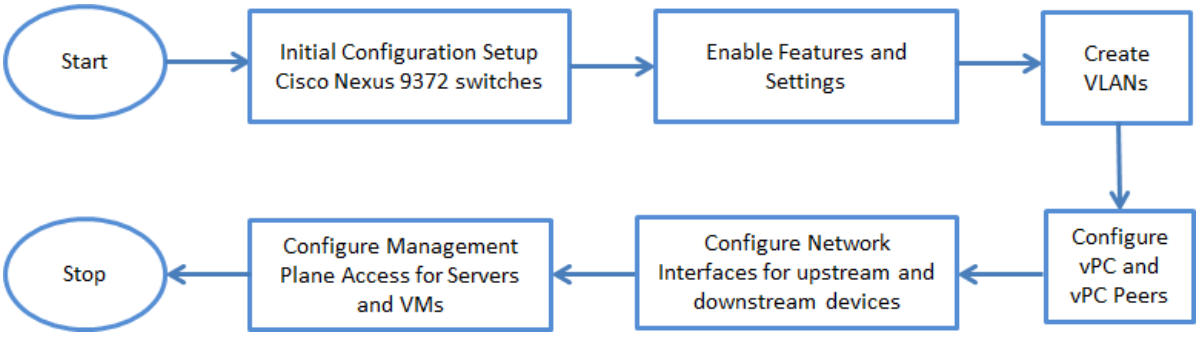


Solution Workflow

This section provides component wise configuration workflow for the solution.

Cisco Nexus 9372PX Configuration

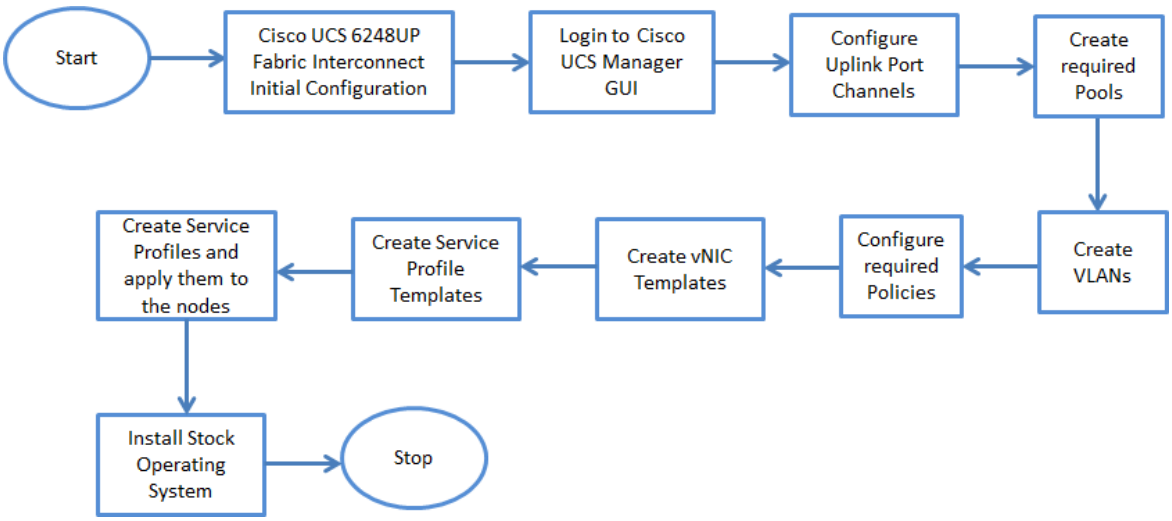
The following workflow shows the high-level configuration steps for Cisco Nexus ToR switches used for north-bound connectivity to the UCSM domain. Application container requiring external data plane access goes through these ToR switches. The SVI for the tenant VLAN for external access is configured on these switches with L3 feature enablement.



For configuration details, refer to the Cisco Nexus 9000 Series switches configuration guides: http://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/products-installation-andconfiguration-guides-list.html

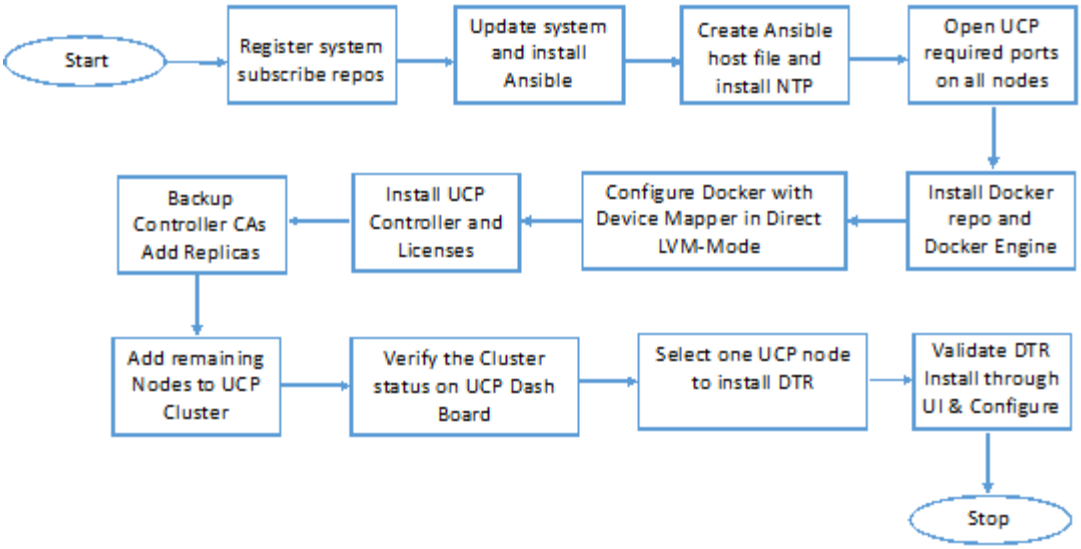
Cisco UCS Manager Configuration

Cisco UCS Manager provides policy based infrastructure management for compute nodes provisioned for various services in this solution. Cisco UCSM is responsible for entire hardware life-cycle management through user defined policy implementation on the compute nodes. Some of the key features include Firmware Management, Faults/Alerts/Stats reporting, Maintenance scheduling, Identity management, Local Disk configuration, Network connectivity and QoS. The compute node hardware characteristics are applied to physical Blade servers through logical configuration known as Service Profile. This logical entity is a collection of various UCSM managed pools and policies to enable compute nodes to utilize hardware components optimally. The concept of stateless computing facilitates much greater scalability and greatly reduces application downtime in the event of hardware failure. The following workflow shows high-level tasks to bring up an UCS compute node. This also enables administrators to quickly provision the hardware resources in a short period of time and drastically reduces the application container bring up time.



Docker Datacenter Installation

Host setup day zero automation tasks are run on all the participating server nodes in the cluster using Ansible automation tool. This tool does not use a client/server model and does not need an additional build server. Any node can be initiated to run minimal host setup tasks even before the Docker datacenter is deployed on the nodes.



Software and Hardware Versions

The following tables provide software and hardware versions used in this solution for both the architectures.

Table 7 lists the Cisco UCS infrastructure components used in the solution for production deployment.

Table 7 Solution Component Details for Production Deployment

Component	Model	Quantity	Comments

UCP Controller, UCP and DTR nodes	Cisco UCS B200 M4 Servers	10	CPU – 2 x E5-2660 V4 Memory – 8 x 16GB 2133 DIMM – total of 128G Local Disks – 2 x 300 GB SAS disks for OS Boot & Docker Engine Network Card – 1x1340 VIC Raid Controller – Cisco MRAID 12 G SAS Controller
Chassis	Cisco UCS 5108 Chassis	2	
IO Modules	Cisco UCS 2208XP Fabric Extenders	4	
Fabric Interconnects	Cisco UCS 6248UP Fabric Interconnects	2	
ToR Switches	Cisco Nexus 9372PX Switches	2	

Table 8 lists the hardware and software versions used in the solution for Production deployment.

Table 8 Hardware and Software Versions for Production Deployment

Layer	Device	Image	Comments
Computing	Cisco UCS B200M4 Servers	Version 3.1 (2c)	Cisco UCS server
Network Adapter	Cisco UCS 1340 Virtual Interface Card (VIC)	Version 3.1 (2c)	Cisco VIC firmware
Network	Cisco UCS 6248UP Fabric Interconnects	Version 3.1 (2c)	Cisco UCS Fabric Interconnect
	Cisco Nexus 9372PX Switches	Version 7.0(3)I1(3)	Cisco N9K TOR Switch
Cisco Software	Cisco UCS Manager	Version 3.1 (2c)	Cisco UCS Manager
Docker Datacenter	Docker CS Engine	Version 1.12.3-cs4	Docker Commercially Supported Engine Note: This solution has been validated on this version of Docker Engine.
	Docker Swarm	Version 1.2.5	Docker Swarm Scheduler is embedded in UCP Note: Docker Swarm version is appropriately

			picked during the UCP install.
	Docker Universal Control Plane (UCP)	Version 1.1.4	Docker Environment Orchestrator and Management Interface Note: This solution has been validated on this version of Docker UCP.
	Docker Trusted Repository (DTR)	Version 2.0.3	Docker Image Store for Enterprise Note: This solution has been validated on this version of DTR
Operating System (OS)	Red Hat Enterprise Linux	Version 7.2	Red Hat Linux for Baremetal OS
NIC Driver	Cisco UCS 1340 Virtual Interface Card (VIC)	Version 2.3.0.30	Cisco eNIC device driver for RHEL 7.2 OS

Table 9 lists the Cisco UCS infrastructure components used in the solution for second architecture.

Table 9 Solution Component Details for Second Architecture

Component	Model	Quantity	Comments
UCP Controller, UCP and DTR nodes	Cisco UCS C220 M4 Servers	4	CPU – 2 x E5-2669 V4 Memory – 16 x 16GB 2133 DIMM – total of 256GB Local Disks – 6 x 1.2 TB SAS disks for OS Boot and Docker Engine Network Card – 1x1227 VIC Raid Controller – Cisco MRAID SAS Controller
Fabric Interconnects	Cisco UCS 6248UP Fabric Interconnects	2	
ToR Switches	Cisco Nexus 9372PX Switches	2	

Table 10 lists the hardware and software versions used in the solution for second architecture.

Table 10 Hardware and Software Versions for Second Architecture

Layer	Device	Image	Comments
Computing	Cisco UCS C220M4 Servers	Version 3.1 (2c)	Cisco UCS server
Network Adapter	Cisco UCS 1227 Virtual Interface Card (VIC)	Version 3.1 (2c)	Cisco VIC firmware
Network	Cisco UCS 6248UP Fabric Interconnects	Version 3.1 (2c)	Cisco UCS Fabric Interconnect
	Cisco Nexus 9372PX Switches	Version 7.0(3)I1(3)	Cisco N9K TOR Switch
Cisco Software	Cisco UCS Manager	Version 3.1 (2c)	Cisco UCS Manager
Docker Datacenter	Docker CS Engine	Version 1.12.3-cs4	Docker Commercially Supported Engine Note: This solution has been validated on this version of Docker Engine.
	Docker Swarm	Version 1.2.5	Docker Swarm Scheduler is embedded in UCP Note: Docker Swarm version is appropriately picked during the UCP install.
	Docker Universal Control Plane (UCP)	Version 1.1.4	Docker Environment Orchestrator and Management Interface Note: This solution has been validated on this version of Docker UCP.
	Docker Trusted Repository (DTR)	Version 2.0.3	Docker Image Store for Enterprise Note: This solution has been validated on this version of DTR
Operating System (OS)	Red Hat Enterprise Linux	Version 7.2	Red Hat Linux for Baremetal OS
NIC Driver	Cisco UCS 1227 Virtual Interface Card (VIC)	Version 2.3.0.30	Cisco eNIC device driver for RHEL 7.2 OS

Solution Deployment

Cisco Nexus 9372PX

Initial Configuration and Setup

This section outlines the initial configuration necessary for bringing up a new Cisco Nexus 9000.

Cisco Nexus A

To set up the initial configuration for the first Cisco Nexus switch complete the following steps:

1. Connect to the serial or console port of the switch

```
Enter the configuration method: console
Abort Auto Provisioning and continue with normal setup? (yes/no[n]: y

---- System Admin Account Setup ----
Do you want to enforce secure password standard (yes/no[y] :
Enter the password for "admin":
Confirm the password for "admin":

---- Basic System Configuration Dialog VDC: 1 ----
This setup utility will guide you through the basic configuration of the system. Setup
configures only enough connectivity for management of the system.
Please register Cisco Nexus9000 Family devices promptly with your supplier. Failure to
register may affect response times for initial service calls. Nexus9000 devices must be
registered to receive entitled support services.
Press Enter at anytime to skip a dialog. Use ctrl-c at anytime to skip the remaining
dialogs.
Would you like to enter the basic configuration dialog (yes/no): y
Create another login account (yes/no) [n]: n
Configure read-only SNMP community string (yes/no) [n]:
Configure read-write SNMP community string (yes/no) [n]:
Enter the switch name: Docker-N9K-A
Continue with Out-of-band (mgmt0) management configuration? (yes/no) [y]:
Mgmt0 IPv4 address: 10.65.121.54
Mgmt0 IPv4 netmask: 255.255.255.0
 Configure the default gateway? (yes/no) [y]:
IPv4 address of the default gateway: 192.168.155.1
Configure advanced IP options? (yes/no) [n]:
Enable the telnet service? (yes/no) [n]:
Enable the ssh service? (yes/no) [y]:
Type of ssh key you would like to generate (dsa/rsa) [rsa]:
Number of rsa key bits <1024-2048> [1024]: 2048
Configure the ntp server? (yes/no) [n]: y
NTP server IPv4 address: 10.65.121.54
Configure default interface layer (L3/L2) [L2]:
Configure default switchport interface state (shut/noshut) [noshut]:
Configure CoPP system profile (strict/moderate/lenient/dense/skip) [strict]:
```

15. Review the settings printed to the console. If they are correct, answer yes to apply and save the configuration

16. Wait for the login prompt to make sure that the configuration has been saved prior to proceeding.

Cisco Nexus B

To set up the initial configuration for the second Cisco Nexus switch complete the following steps:

1. Connect to the serial or console port of the switch
2. The Cisco Nexus B switch should present a configuration dialog identical to that of Cisco Nexus A shown above. Provide the configuration parameters specific to Cisco Nexus B for the following configuration variables. All other parameters should be identical to that of Cisco Nexus A.

   - `Admin password`
   - `Nexus B Hostname: Docker-N9K-B`
   - `Nexus B mgmt0 IP address: 10.65.121.55`
   - `Nexus B mgmt0 Netmask: 255.255.255.0`
   - `Nexus B mgmt0 Default Gateway: 192.168.155.1`

Feature Enablement

The following commands enable the IP switching feature and set default spanning tree behaviors:

3. On each Nexus 9000, enter the configuration mode:

   ```
   config terminal
   ```

4. Use the following commands to enable the necessary features:

   ```
   feature udld

   feature lacp

   feature vpc

   feature interface-vlan
   ```

5. Configure the spanning tree and save the running configuration to start-up:

   ```
   spanning-tree port type network default

   spanning-tree port type edge bpduguard default

   spanning-tree port type edge bpdufilter default

   copy run start
   ```

vLAN Creation

To create the necessary virtual local area networks (VLANs), complete the following step on both switches:

From the configuration mode, run the following commands:

```
vlan 603

name vlan603
```

Configure vPC

Configuring vPC Domain

Cisco Nexus A

To configure virtual port channels (vPCs) for switch A, complete the following steps:

1. From the global configuration mode, create a new vPC domain:

   ```
   vpc domain 10
   ```

2. Make Cisco Nexus A the primary vPC peer by defining a low priority value:

   ```
   role priority 10
   ```

3. Use the management interfaces on the supervisors of the Cisco Nexus switches to establish a keepalive link:

   ```
   peer-keepalive destination 10.65.121.55 source 10.65.121.54
   ```

4. Enable following features for this vPC domain:

   ```
   peer-switch
   delay restore 150
   peer-gateway
   ip arp synchronize
   auto-recovery
   ```

5. Save the configuration.

   ```
   copy run start
   ```

Cisco Nexus B

To configure vPCs for switch B, complete the following steps:

1. From the global configuration mode, create a new vPC domain:

   ```
   vpc domain 10
   ```

2. Make Cisco Nexus A the primary vPC peer by defining a higher priority value on this switch:

   ```
   role priority 20
   ```

3. Use the management interfaces on the supervisors of the Cisco Nexus switches to establish a keepalive link:

   ```
   peer-keepalive destination 10.65.121.54 source 10.65.121.55
   ```

4. Enable following features for this vPC domain:

   ```
   peer-switch
   delay restore 150
   peer-gateway
   ip arp synchronize
   auto-recovery
   ```

5. Save the configuration:

   ```
   copy run start
   ```

Configuring Network Interfaces for VPC Peer Links

Cisco Nexus A

1. Define a port description for the interfaces connecting to VPC Peer Docker-N9K-B.

   ```
   interface Eth1/9
   description VPC Peer Docker-N9K-B:e1/10
   interface Eth1/10
   description VPC Peer Docker-N9K-B:e1/9
   ```

2. Apply a port channel to both VPC Peer links and bring up the interfaces.

```
interface Eth1/9,Eth1/10
channel-group 11 mode active
no shutdown
```

3. Enable UDLD on both interfaces to detect unidirectional links.

```
udld enable
```

4. Define a description for the port-channel connecting to Docker-N9K-B.

```
interface port-channel 11
description vPC peer-link
```

5. Make the port-channel a switchport, and configure a trunk to allow in-band management, VM traffic, and the native VLAN.

```
switchport
switchport mode trunk
switchport trunk native vlan 603
spanning-tree port type network
```

6. Make this port-channel the VPC peer link and bring it up.

```
vpc peer-link
no shutdown
copy run start
```

Cisco Nexus B

1. Define a port description for the interfaces connecting to VPC Peer Docker-N9K-A.

```
interface Eth1/9
description VPC Peer Docker-N9K-A:e1/10
interface Eth1/10
description VPC Peer Docker-N9K-A:e1/9
```

2. Apply a port channel to both VPC Peer links and bring up the interfaces.

```
interface Eth1/9,Eth1/10
channel-group 11 mode active
no shutdown
```

3. Enable UDLD on both interfaces to detect unidirectional links.

```
udld enable
```

4. Define a description for the port-channel connecting to Docker-N9K-A.

```
interface port-channel 11
description vPC peer-link
```

5. Make the port-channel a switchport, and configure a trunk to allow in-band management, VM traffic, and the native VLAN.

```
switchport
switchport mode trunk
switchport trunk native vlan 603
spanning-tree port type network
```

6. Make this port-channel the VPC peer link and bring it up.

```
vpc peer-link
no shutdown
copy run start
```

Configure Network Interfaces

Cisco Nexus A

1. Define a description for the port-channel connecting to Docker-FI-A.

```
interface port-channel 12
description Docker-FI-A
```

2. Make the port-channel a switchport, and configure a trunk to allow in-band management, VM traffic, and the native VLANs.

```
switchport mode trunk
switchport trunk native vlan 603
```

3. Make the port channel and associated interfaces spanning tree edge ports.

```
spanning-tree port type edge trunk
spanning-tree guard root
no lacp graceful-convergence
```

4. Make this a VPC port-channel and bring it up.

```
vpc 12
no shutdown
```

5. Define a port description for the interface connecting to Docker-FI-A.

```
interface Eth1/11
```

6. Apply it to a port channel and bring up the interface.

```
channel-group 12 mode active
no shutdown
```

7. Enable UDLD to detect unidirectional links.

```
udld enable
```

8. Define a description for the port-channel connecting to Docker-FI-B.

```
interface port-channel
description Docker-FI-B
```

9. Make the port-channel a switchport, and configure a trunk to allow in-band management, VM traffic VLANs and the native VLAN.

```
switchport mode trunk
switchport trunk native vlan 603
```

10. Make the port channel and associated interfaces spanning tree edge ports.

```
spanning-tree port type edge trunk
spanning-tree guard root
no lacp graceful-convergence
```

11. Make this a VPC port-channel and bring it up.

```
vpc 13
no shutdown
```

12. Define a port description for the interface connecting to Docker-FI-B.

```
interface Eth1/12
```

13. Apply it to a port channel and bring up the interface.

```
channel-group 13 mode active
no shutdown
```

14. Enable UDLD to detect unidirectional links.

```
udld enable

copy run start
```

Cisco Nexus B

1. Define a description for the port-channel connecting to Docker-FI-B.

```
interface port-channel 12
description Docker-FI-B
```

2. Make the port-channel a switchport, and configure a trunk to allow in-band management, VM traffic, and the native VLANs.

```
switchport mode trunk
switchport trunk native vlan 603
```

3. Make the port channel and associated interfaces spanning tree edge ports.

```
spanning-tree port type edge trunk
spanning-tree guard root
no lacp graceful-convergence
```

4. Make this a VPC port-channel and bring it up.

```
vpc 12
no shutdown
```

5. Define a port description for the interface connecting to Docker-FI-B.

```
interface Eth1/11
```

6. Apply it to a port channel and bring up the interface.

```
channel-group 12 mode active
no shutdown
```

7. Enable UDLD to detect unidirectional links.

```
udld enable
```

8. Define a description for the port-channel connecting to Docker-FI-A.

```
interface port-channel 13
description Docker-FI-A
```

9. Make the port-channel a switchport, and configure a trunk to allow in-band management, and VM traffic VLANs and the native VLAN.

```
switchport mode trunk
switchport trunk native vlan 603
```

10. Make the port channel and associated interfaces spanning tree edge ports.

```
spanning-tree port type edge trunk
spanning-tree guard root
no lacp graceful-convergence
```

11. Make this a VPC port-channel and bring it up.

```
vpc 13
no shutdown
```

12. Define a port description for the interface connecting to Docker-N9K-A.

```
interface Eth1/12
```

13. Apply it to a port channel and bring up the interface.

```
channel-group 13 mode active
no shutdown
```

14. Enable UDLD to detect unidirectional links.

```
udld enable

copy run start
```

Cisco UCS Manager

Initial Setup of Cisco Fabric Interconnects

A pair of Cisco UCS 6248UP Fabric Interconnects is used in this design. The minimum configuration required for bringing up the FIs and the embedded Cisco UCS Manager (UCSM) is outlined below. All configurations after this will be done using Cisco UCS Manager.

Cisco UCS 6248UP FI – Primary (FI-A)

1. Connect to the console port of the primary Cisco UCS FI.

```
Enter the configuration method: console
Enter the setup mode; setup newly or restore from backup.(setup/restore)? Setup You
have chosen to setup a new fabric interconnect? Continue? (y/n): y
Enforce strong passwords? (y/n) [y]: y
Enter the password for "admin": <Enter Password>
Enter the same password for "admin": <Enter Password>
Is this fabric interconnect part of a cluster (select 'no' for standalone)? (yes/no)
[n]: y
Which switch fabric (A|B): A
Enter the system name: Docker-FI
Physical switch Mgmt0 IPv4 address: 10.65.122.130
Physical switch Mgmt0 IPv4 netmask: 255.255.255.0
IPv4 address of the default gateway: 10.65.122.1
Cluster IPv4 address: 10.65.122.132
Configure DNS Server IPv4 address? (yes/no) [no]: y
DNS IPv4 address: 171.70.168.183
Configure the default domain name? y
Default domain name: <domain name>
Join centralized management environment (UCS Central)? (yes/no) [n]: <Enter>
```

2. Review the settings printed to the console. If they are correct, answer yes to apply and save the configuration.
3. Wait for the login prompt to make sure that the configuration has been saved prior to proceeding.

Cisco UCS 6248UP FI – Secondary (FI-B)

1. Connect to the console port on the second FI on Cisco UCS 6248UP FI.

```
Enter the configuration method: console
Installer has detected the presence of a peer Fabric interconnect. This Fabric inter-
```

```
connect will be added to the cluster.  Do you want to continue {y|n}?   y
Enter the admin password for the peer fabric interconnect: <Enter Password>
Peer Fabric interconnect Mgmt0 IPv4 address: 10.65.122.130

Peer Fabric interconnect Mgmt0 IPv4 netmask: 255.255.255.0

Cluster IPv4 address: 10.65.122.131
Apply and save the configuration (select 'no' if you want to re-enter)?(yes/no): y
```

2. Verify the above configuration by using Secure Shell (SSH) to login to each FI and verify the cluster status. Status should be as follows if the cluster is up and running properly.

```
Docker-FI-A# show cluster state
```

Now you are ready to log into Cisco UCS Manager using either the individual or cluster IPs of the Cisco UCS Fabric Interconnects.

Configure Ports for Server, Network and Storage Access

Logging into Cisco UCS Manager

To log into the Cisco Unified Computing System (UCS) environment, complete the following steps:

1. Open a web browser and navigate to the Cisco UCS 6248 Fabric Interconnect cluster IP address configured in earlier step.
2. Click Launch Cisco UCS Manager link to download the Cisco UCS Manager software.
3. If prompted, accept security certificates as necessary.
4. When prompted, enter admin as the user name and enter the administrative password.
5. Click Login to log in to Cisco UCS Manager.
6. Select Yes or No to authorize Anonymous Reporting if desired and click OK.

Cisco UCS Manager – Synchronize to NTP

To synchronize the Cisco UCS environment to the NTP server, complete the following steps:

1. From Cisco UCS Manager, click Admin tab in the navigation pane.
2. Select All > Timezone Management > Timezone.
3. Right-click and select Add NTP Server.
4. Specify NTP Server IP (for example, 171.68.38.66) and click OK twice to save edits. The Time Zone can also be specified in the Properties section of the Time Zone window.

Upgrading Cisco UCS Manager

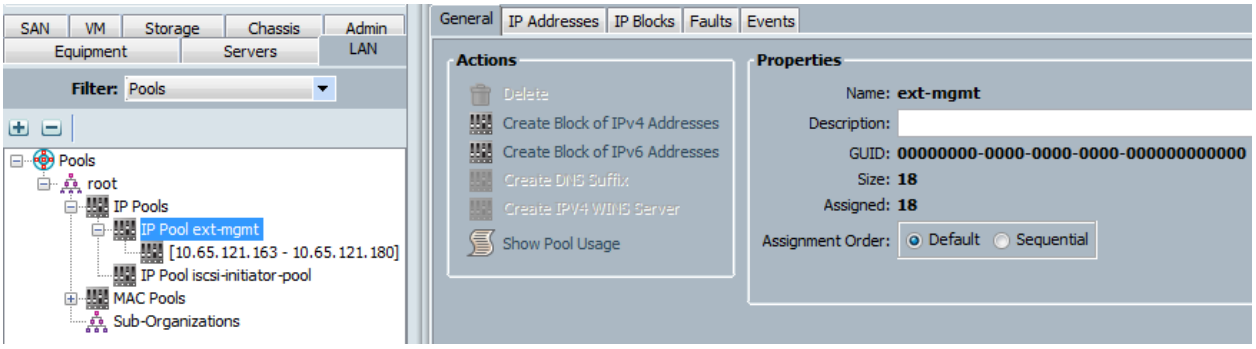
This document assumes that the Cisco UCS Manager is running the version outlined in the Software Matrix. If an upgrade is required, follow the procedures outlined in the Cisco UCS Install and Upgrade Guides.

Assigning Block of IP addresses for KVM Access

To create a block of IP addresses for in-band access to servers in the Cisco UCS environment, complete the following steps. The addresses are used for Keyboard, Video, and Mouse (KVM) access to individual servers managed by Cisco UCS Manager.

This block of IP addresses should be in the same subnet as the management IP addresses for the Cisco UCS Manager.

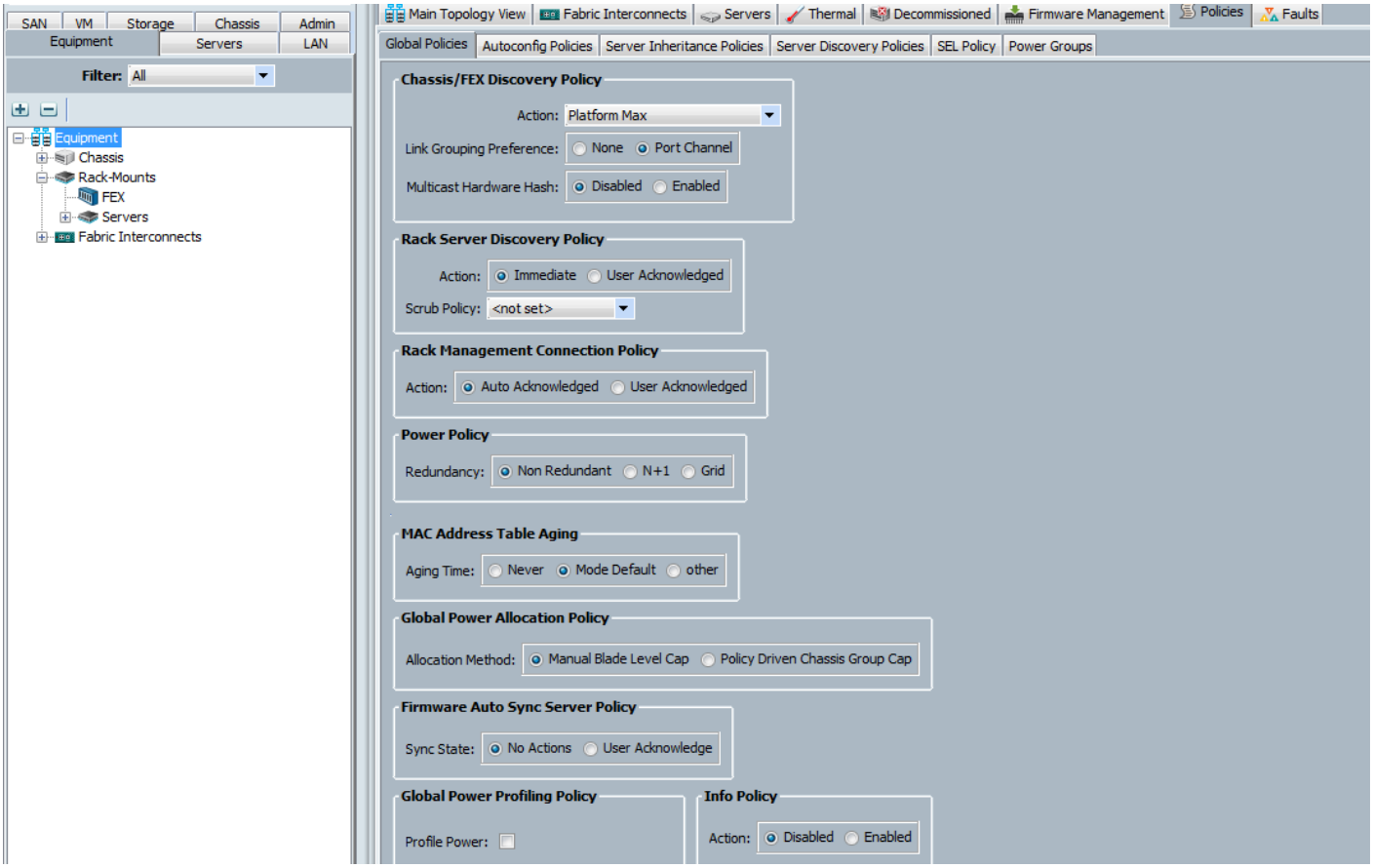
1. From Cisco UCS Manager, click LAN tab in the navigation pane.
2. Select LAN > Pools > root > IP Pools.
3. Right-click and select Create IP Pool.
4. Specify a Name (for example, ext-mgmt) for the pool. Click Next.
5. Click [+] Add to add a new IP Block. Click Next.
6. Enter the starting IP address (From), the number of IP addresses in the block (Size), the Subnet Mask, Default Gateway and DNS information. Click OK.
7. Click Finish to create the IP block.



Editing Chassis Discovery Policy

Setting the discovery policy simplifies the addition of Cisco UCS Blade Server chassis and Cisco Fabric Extenders. To modify the chassis discovery policy, complete the following steps:

1. From Cisco UCS Manager, click Equipment tab in the navigation pane and select Equipment in the list on the left.
2. In the right pane, click Policies tab.
3. Under Global Policies, set the Chassis/FEX Discovery Policy to match the number of uplink ports that are cabled between the chassis or fabric extenders (FEXes) and the fabric interconnects.
4. Set the Link Grouping Preference to Port Channel.
5. Click Save Changes and then OK to complete.



Acknowledging Cisco UCS Chassis

To acknowledge all Cisco UCS chassis, complete the following steps:

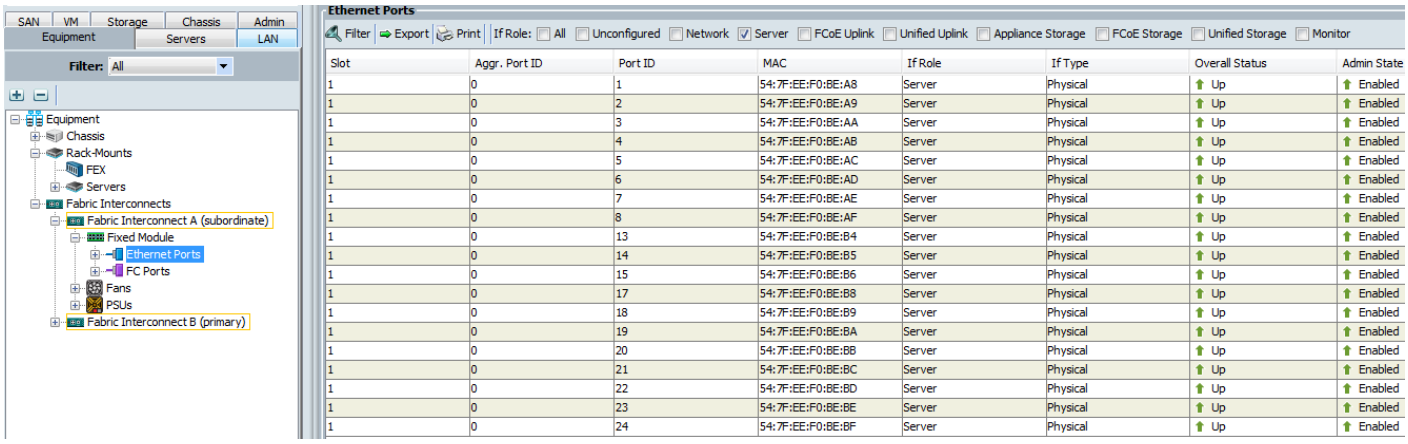
1. From Cisco UCS Manager, click Equipment tab in the navigation pane.
2. Expand Chassis and for each chassis in the deployment, right-click and select Acknowledge Chassis.
3. In the Acknowledge Chassis pop-up, click Yes and then click OK.

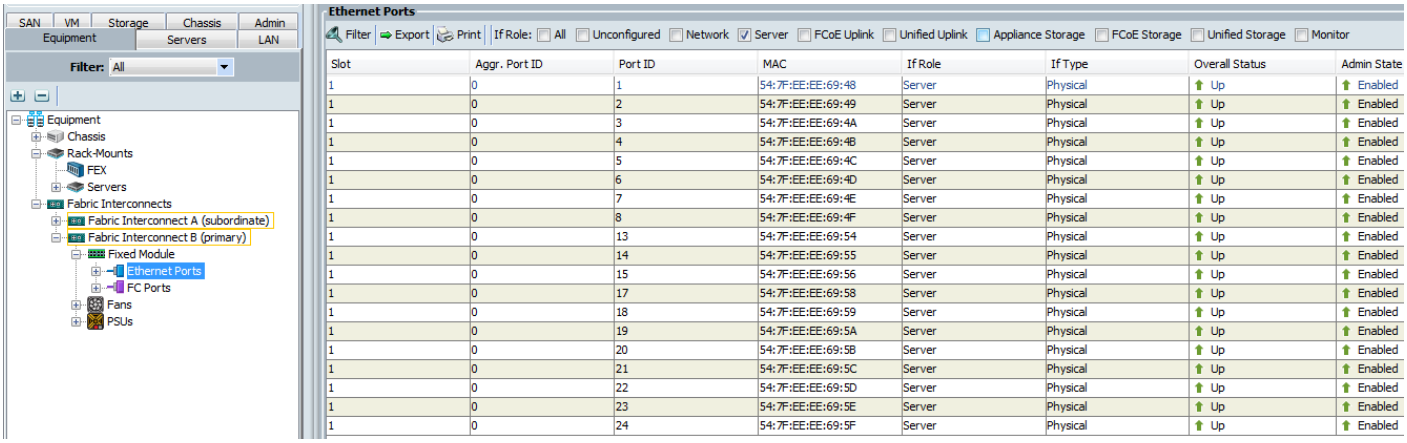
Enabling Server Ports

To configure ports connected to Cisco UCS servers as Server ports, complete the following steps:

1. From Cisco UCS Manager, click Equipment tab in the navigation pane.
2. Select Equipment > Fabric Interconnects > Fabric Interconnect A (primary) > Fixed Module.
3. Expand Ethernet Ports.
4. Select the ports that are connected to Cisco UCS Blade server chassis. Right-click and select Configure as Server Port.
5. Click Yes and then OK to confirm the changes.
6. Repeat above steps for Fabric Interconnect B (secondary) ports that connect to servers.

7. Verify that the ports connected to the servers are now configured as server ports. The view below is filtered to only show Server ports.



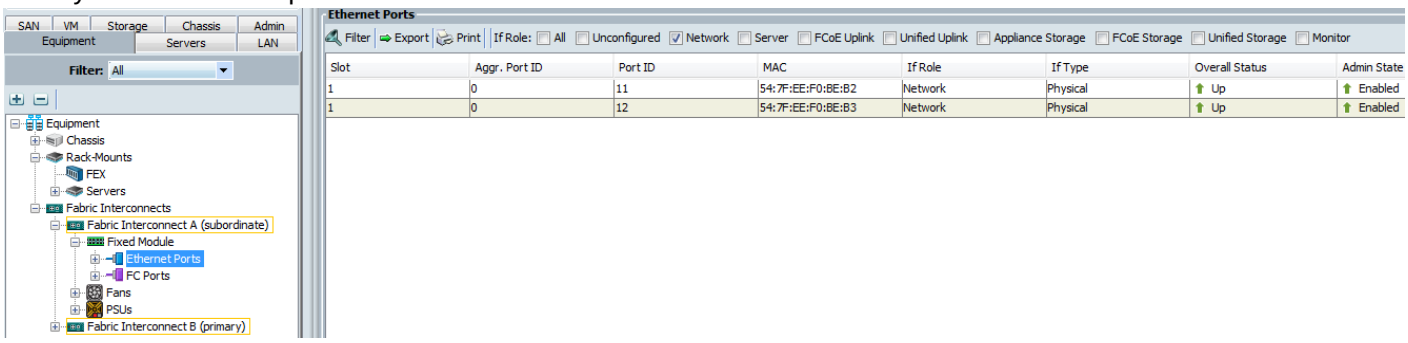


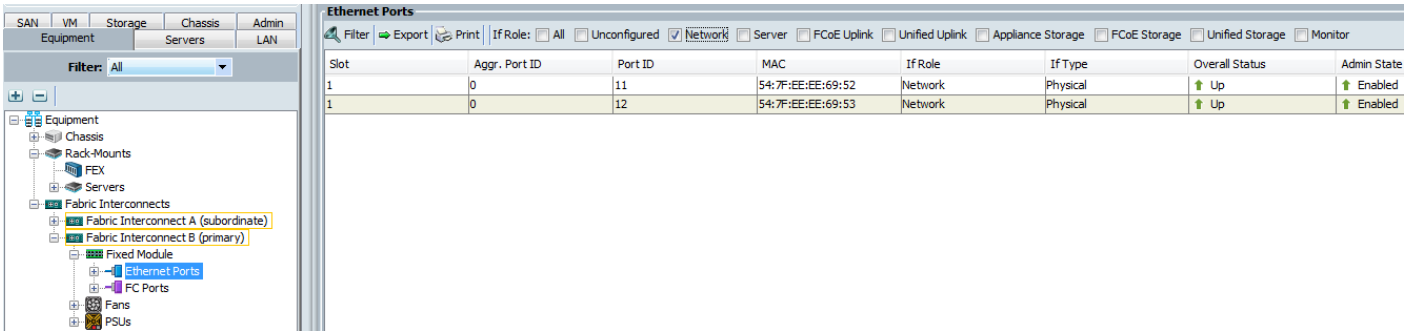
Enabling Uplink Ports to Cisco Nexus 9000 Series Switches

To configure ports connected to Cisco Nexus switches as Network ports, complete the following steps:

1. From Cisco UCS Manager, click Equipment tab in the navigation pane.
2. Select Equipment > Fabric Interconnects > Fabric Interconnect A (primary) > Fixed Module.
3. Expand Ethernet Ports.
4. Select the first port (for example, Port 11) that connects to Cisco Nexus A switch, right-click and select Configure as Uplink Port > Click Yes to confirm the uplink ports and click OK. Repeat for second port (for example, Port 16) that connects to Cisco Nexus B switch.
5. Repeat above steps for Fabric Interconnect B (secondary) uplink ports that connect to Cisco Nexus A and B switches.

Verify that the ports connected to the servers are now configured as server ports. The view below is filtered to only show Network ports.



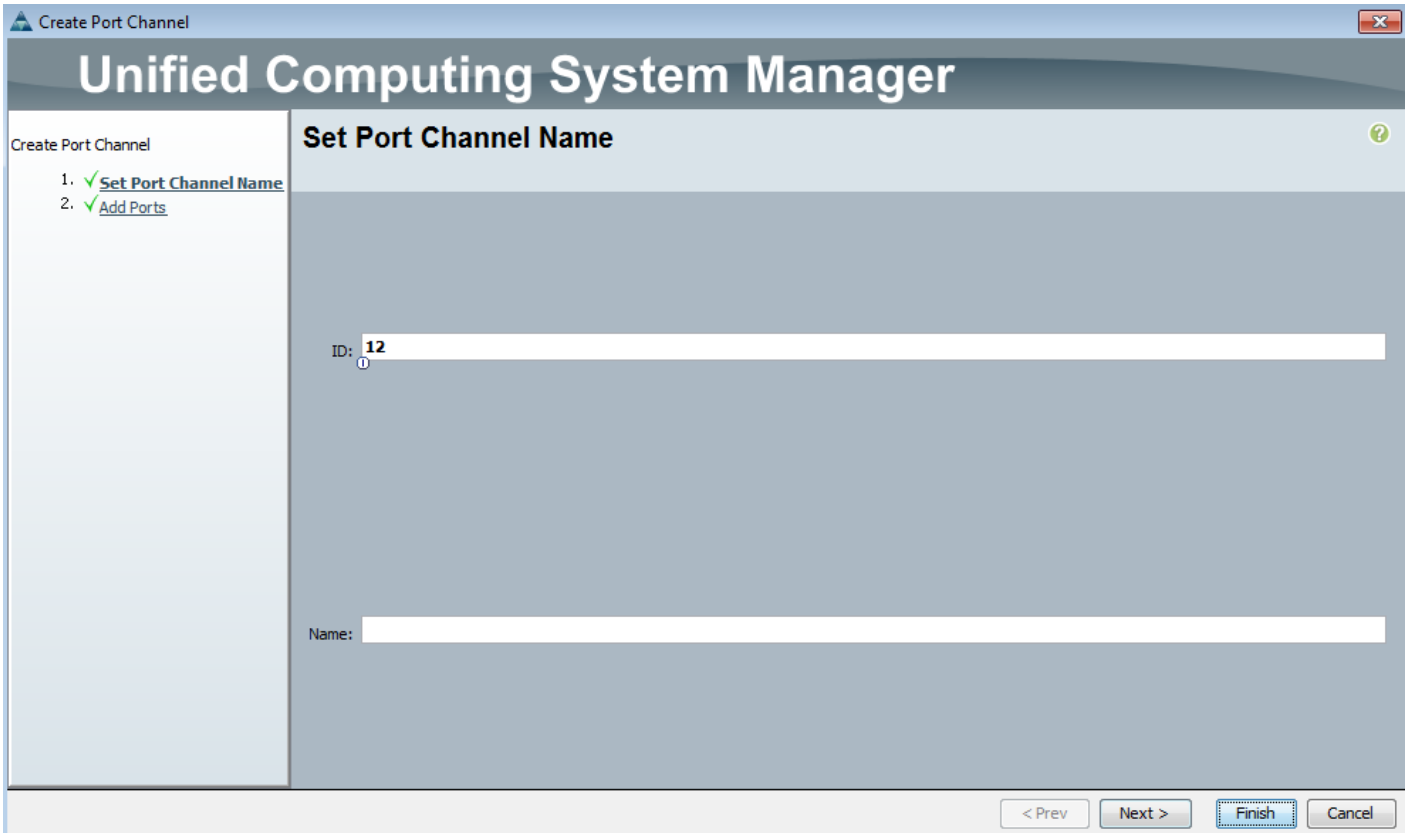


Configuring Port Channels on Uplink Ports to Cisco Nexus 9000 Series Switches

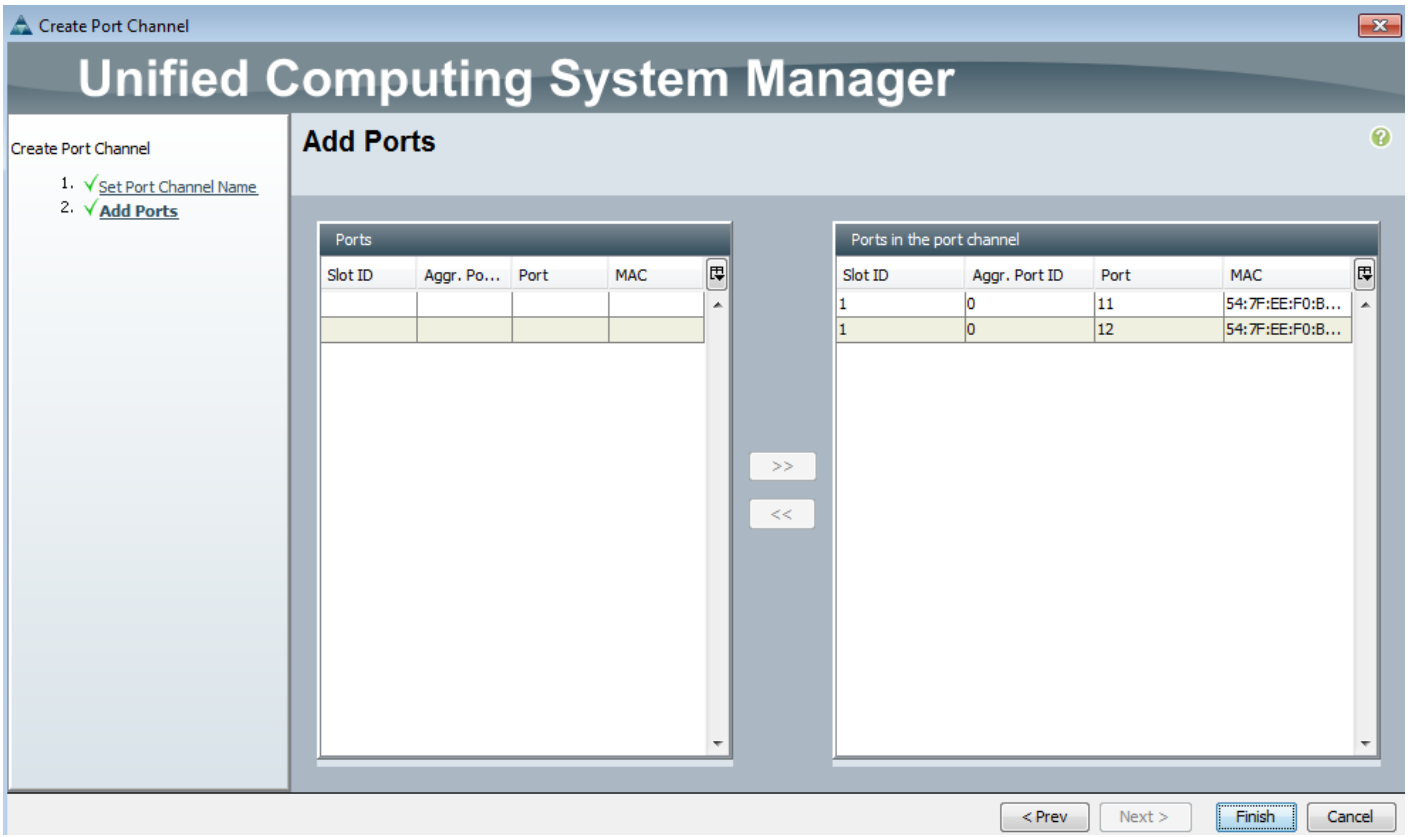
In this procedure, two port channels are created, one from Fabric A to both the Cisco Nexus switches and one from Fabric B to both the Cisco Nexus switches.

To configure port channels on Uplink/Network ports connected to Cisco Nexus switches, complete the following steps:

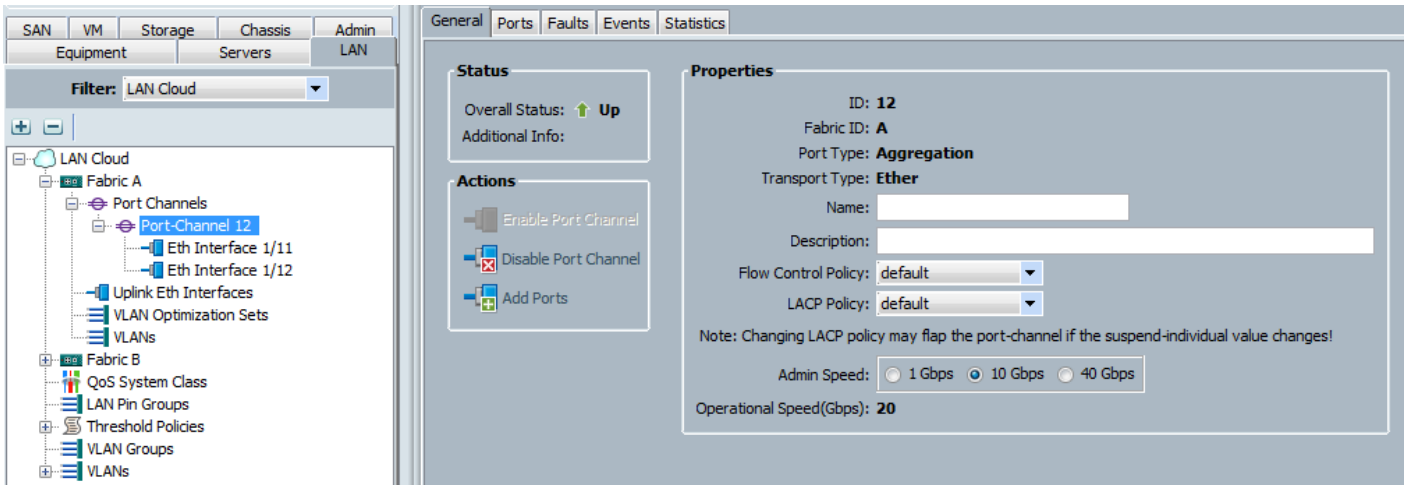
1. From Cisco UCS Manager, click LAN tab in the navigation pane.
2. Select LAN > LAN Cloud > Fabric A > Port Channels.
3. Right-click and select Create Port Channel.
4. In the Create Port Channel window, specify a Name and unique ID.



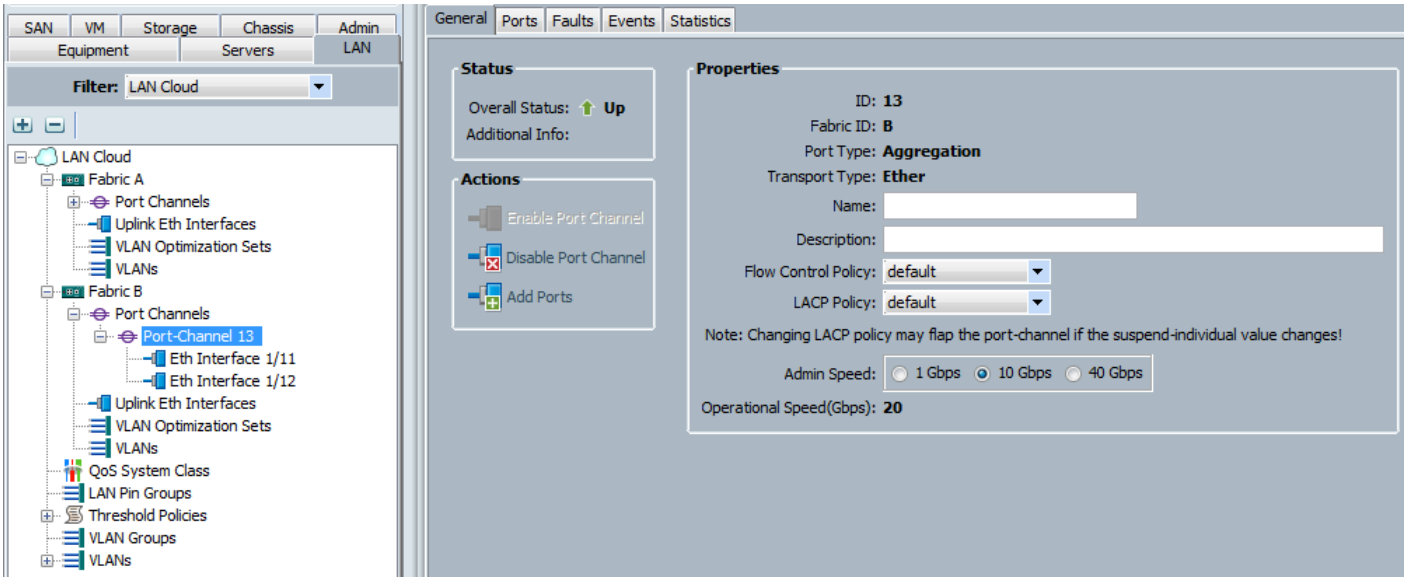
5. In the Create Port Channel window, select the ports to put in the channel (for example, Eth1/11 and Eth1/12). Click Finish to create the port channel.



6. Verify the resulting configuration.



7. Repeat above steps for Fabric B and verify the configuration.

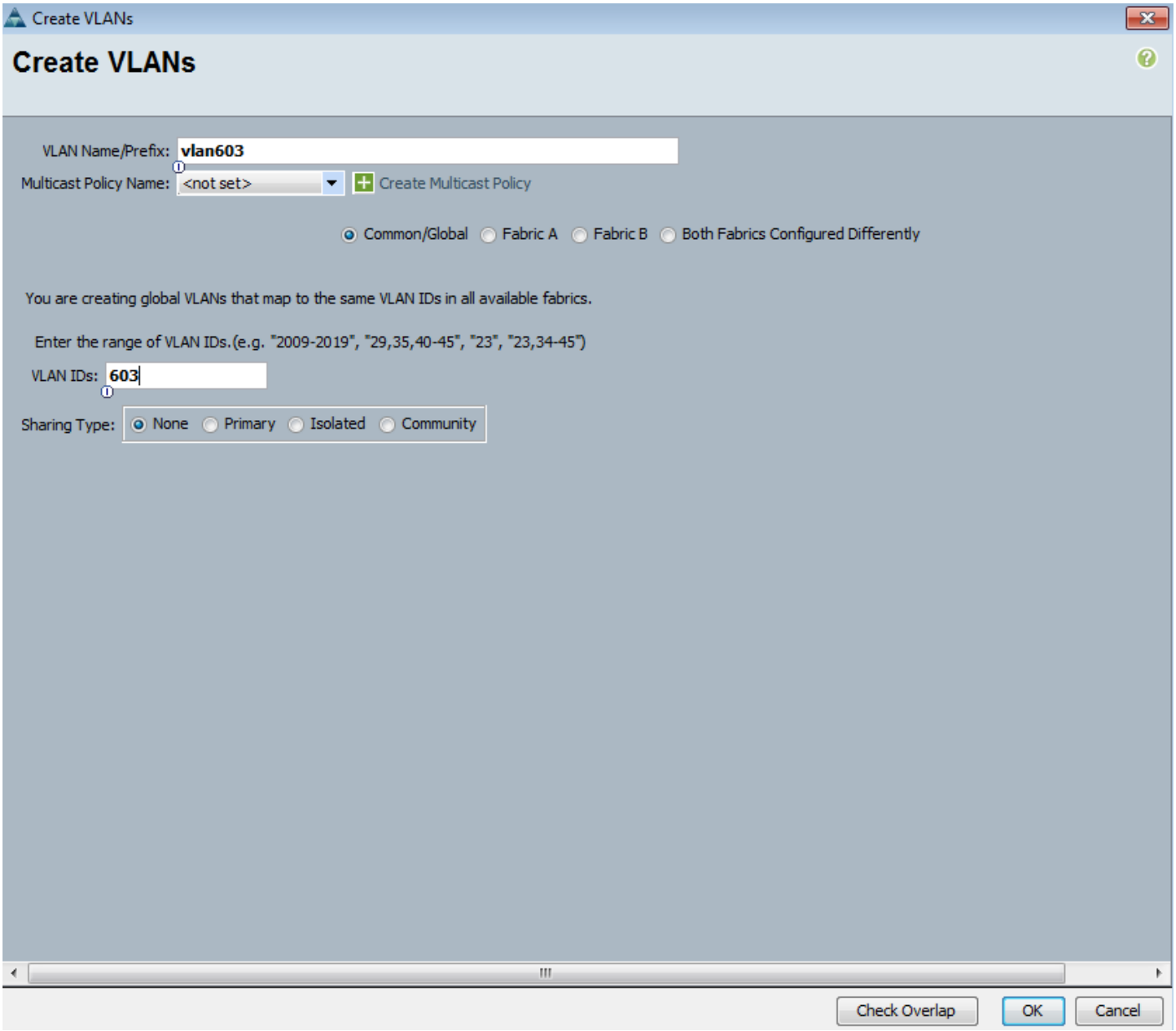


Cisco UCS Configuration – LAN

Creating VLANs

Complete these steps to create necessary VLANs.

1. From Cisco UCS Manager, click LAN tab in the navigation pane.
2. Select LAN > LAN Cloud > VLANs.
3. Right-click and select Create VLANs. Specify a name (for example, 603) and VLAN ID (for example, 603).



4. If the newly created VLAN is a native VLAN, select VLAN, right-click and select Set as Native VLAN from the list. Either option is acceptable, but it needs to match what the upstream switch is set to.

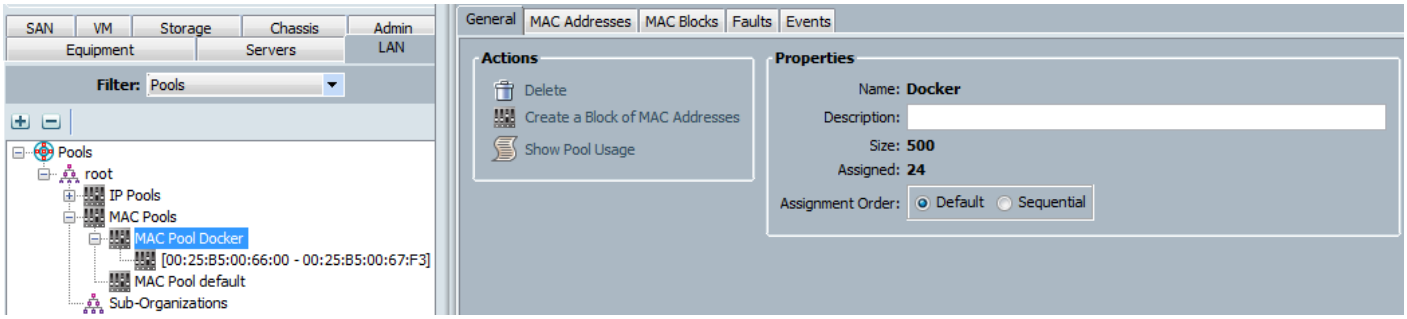
Creating LAN Pools

Creating MAC Address Pools

The MAC addresses in this pool will be used for traffic through Fabric Interconnect A and Fabric Interconnect B.

1. From Cisco UCS Manager, click LAN tab in the navigation pane.
2. Select LAN > Pools > root > MAC Pools.
3. Right-click and select Create Mac Pool.
4. Specify a name (for example, Docker) that identifies this pool.

5. Leave the Assignment Order as Default and click Next.
6. Click [+] Add to add a new MAC pool.
7. For ease-of-troubleshooting, change the 4th and 5th octet to AA:AA traffic using Fabric Interconnect A. Generally speaking, the first three octets of a mac-address should not be changed.
8. Select a size (for example, 24) and select OK and then click Finish to add the MAC pool.



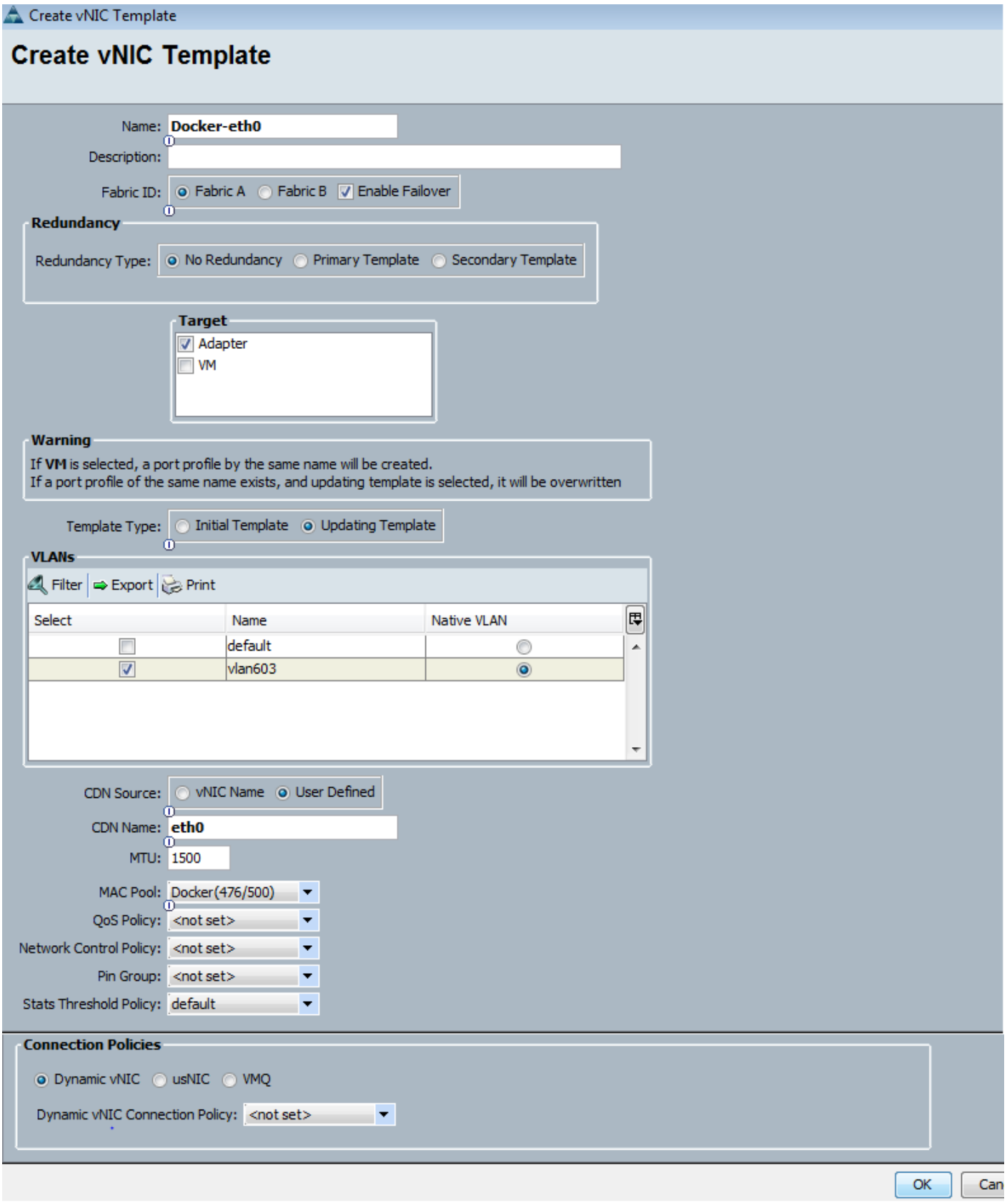
Creating LAN Policies

Creating vNIC Templates

To create virtual network interface card (vNIC) templates for Cisco UCS hosts, complete the following steps. Two vNICs are created for redundancy – one through Fabric A and another through Fabric B. All host traffic is carried across these two vNICs in this design.

Creating vNIC Template for Fabric A

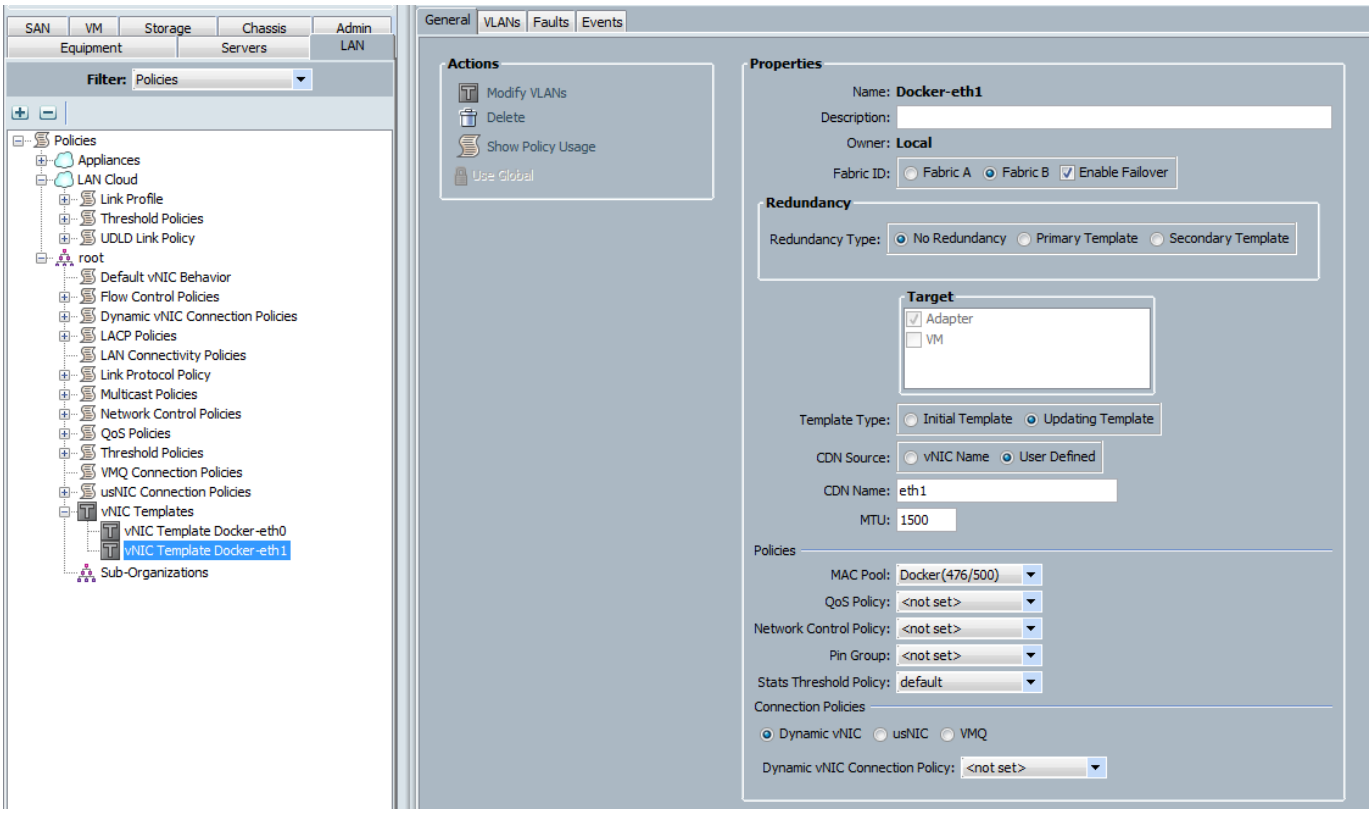
1. From Cisco UCS Manager, select LAN tab in the navigation pane.
2. Select LAN > Policies > root > vNIC Templates.
3. Right-click and select Create vNIC Template.
4. Specify a template Name (for example, Docker-eth0) for the policy.
5. Keep Fabric A selected and keep Enable Failover checkbox checked.
6. Under Target, make sure that the VM checkbox is NOT selected.
7. Select Updating Template as the Template Type.
8. Under VLANs, select the checkboxes for all VLAN traffic that a host needs to see (for example, 603) and select the Native VLAN radio button.
9. For CDN Source, select User Defined radio button. This option ensures that the defined vnic name gets reflected as the adapter's network interface name during OS installation.
10. For CDN Name, enter a suitable name.
11. Keep the MTU as 1500.
12. For MAC Pool, select the previously configured LAN pool (for example, Docker).
13. Choose the default values in the Connection Policies section.



14. Click OK to create the vNIC template.

Creating vNIC Template for Fabric B

Repeat the above steps to create a vNIC template (for example, Docker-eth0) through Fabric B.



Cisco UCS Configuration – Server

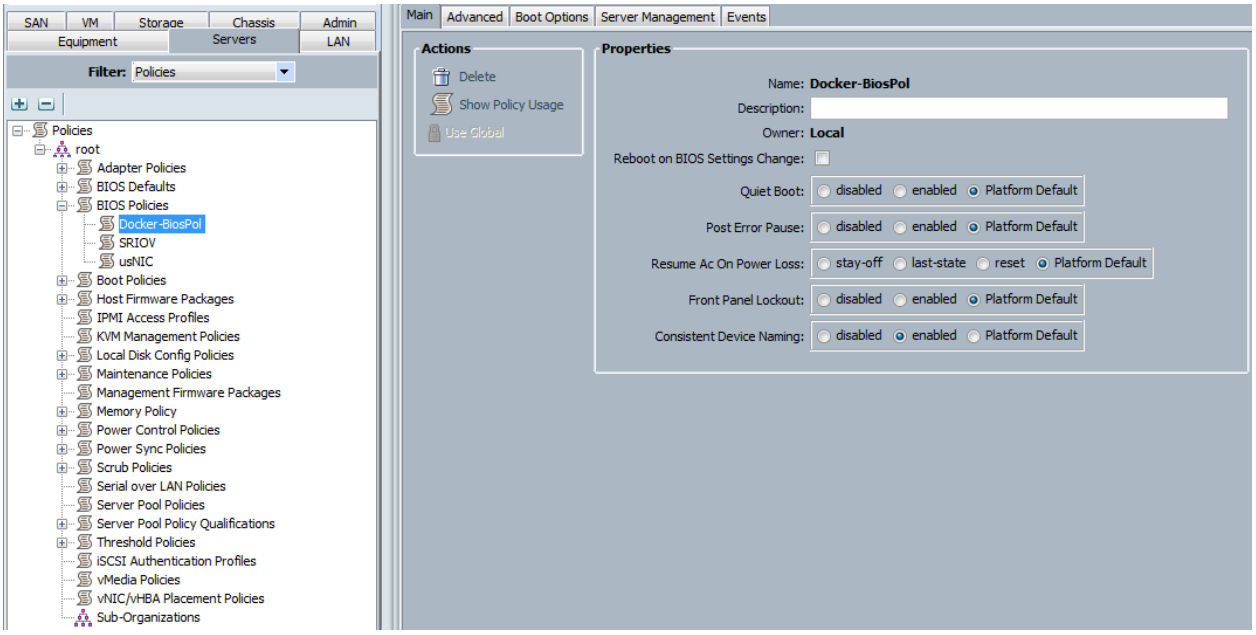
Creating Server Policies

In this procedure, various server policies that are used in this solution are created.

Creating BIOS Policy

To create a server BIOS policy for Cisco UCS hosts, complete the following steps:

1. In Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Policies > root > BIOS Policies.
3. Right-click and select Create BIOS Policy.
4. In the Main screen, enter BIOS Policy Name (for example, Docker-BiosPol) and change the Consistent Device Naming to enabled. Click Next.

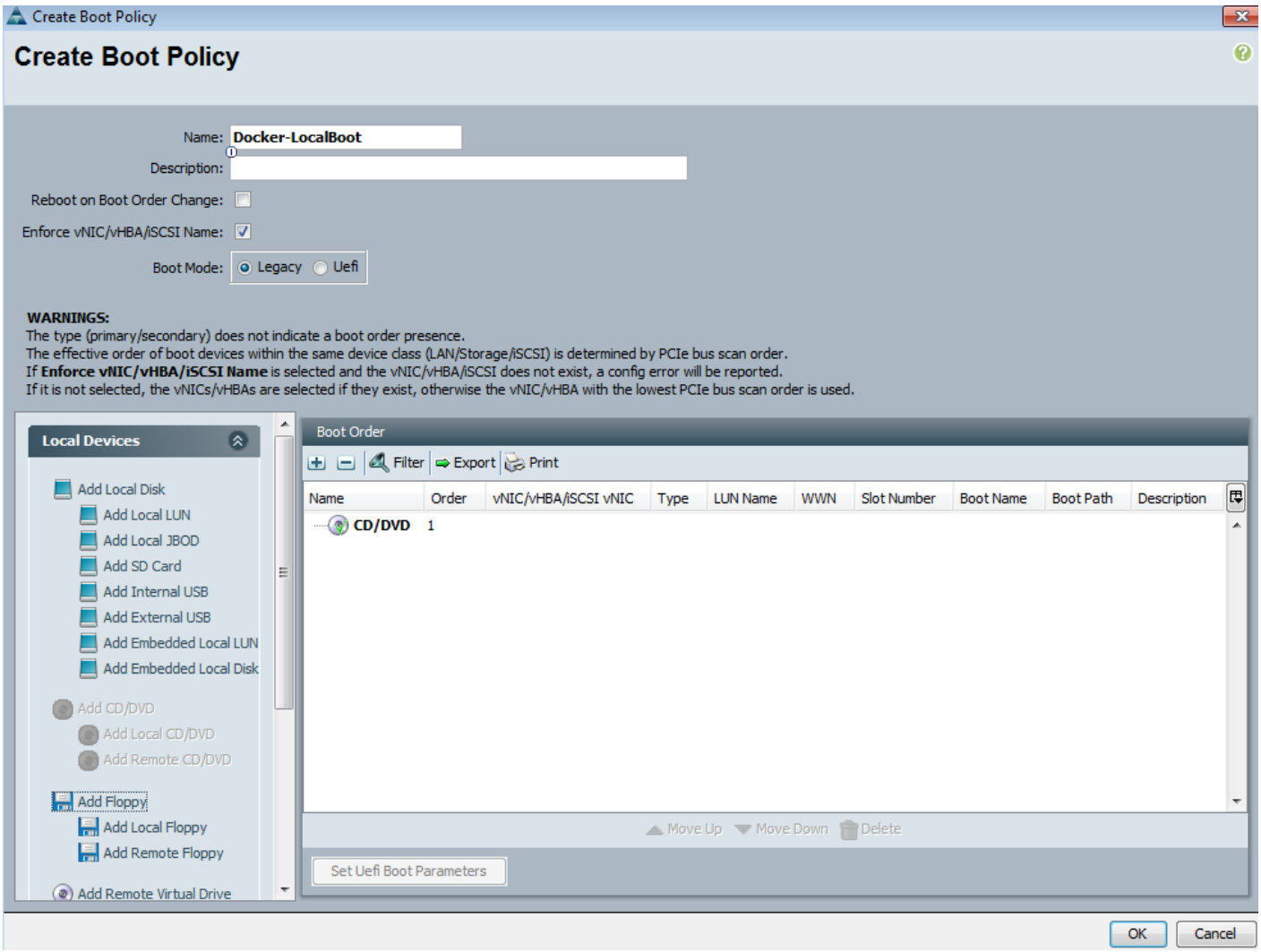


5. Keep the other options in all the other tabs at Platform Default.
6. Click Finish and OK to create the BIOS policy.

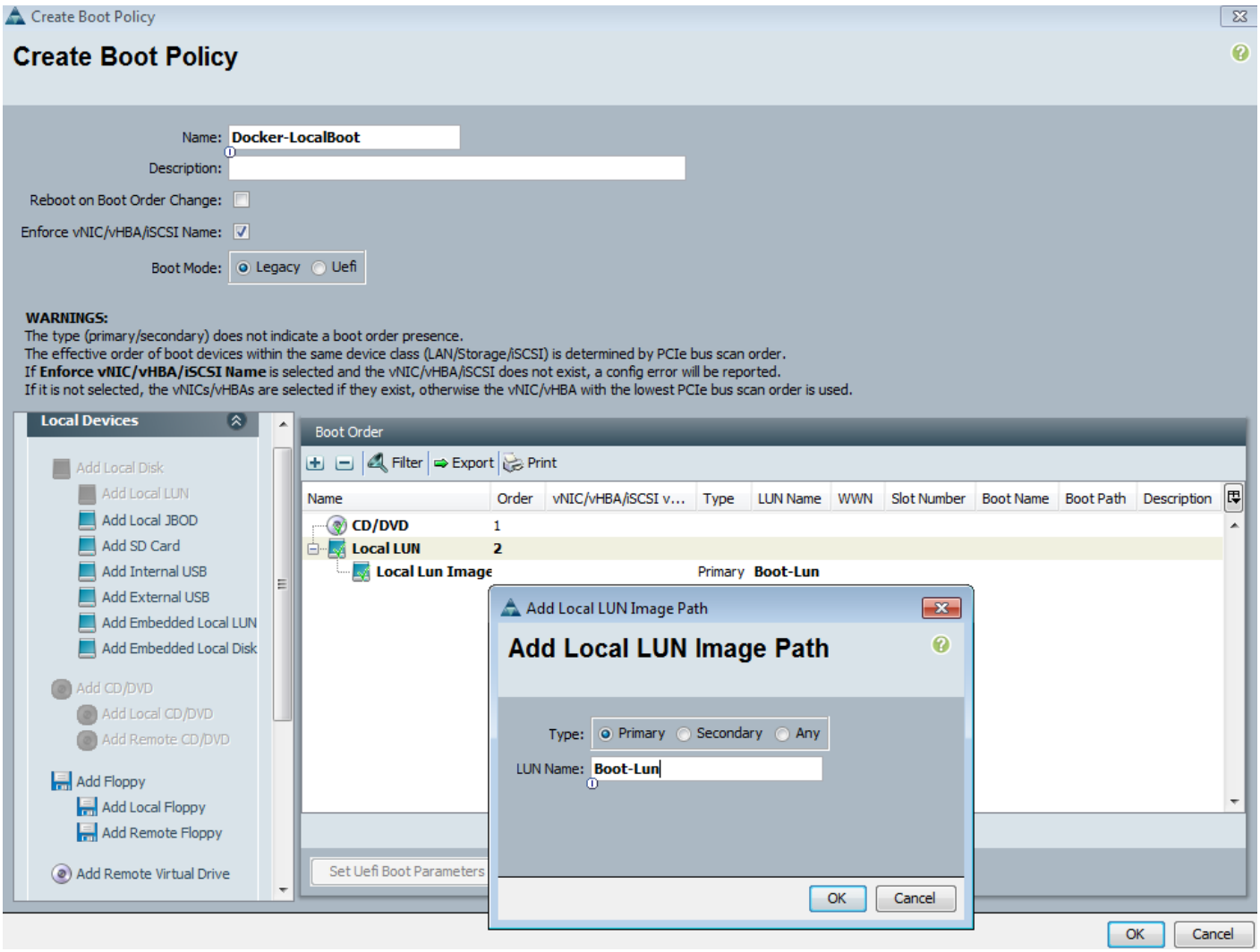
Creating Boot Policy

To create the boot policy, complete the following steps:

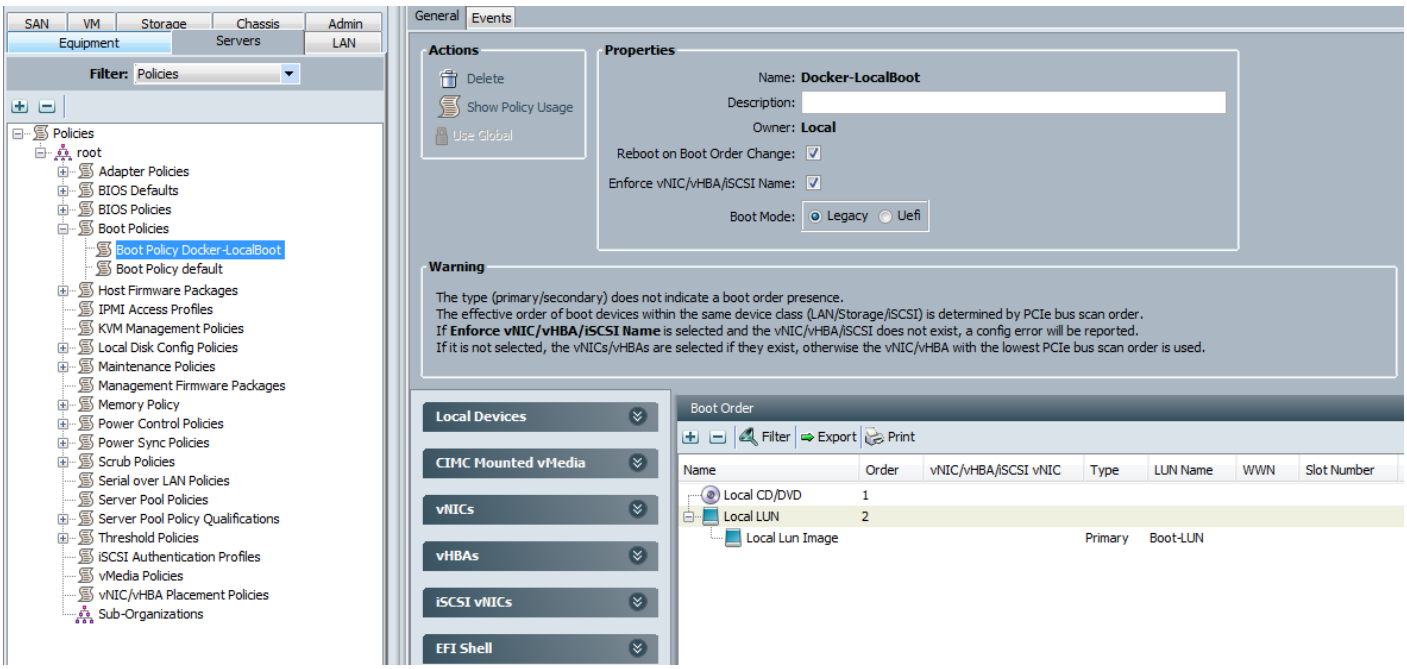
1. In Cisco UCS Manager, click the Servers tab in the navigation pane.
2. Select Policies > root > Boot Policies.
3. Right-click and select Create Boot Policy.
4. In the Create Boot Policy window, enter the policy name (for example, Docker-LocalBoot).
5. Expand the Local Devices section of the window and select Add CD/DVD. The Local CD/DVD and Remote CD/DVD will get greyed out.



6. Now select Add Local Lun under Add Local Disk. In the pop-up window select Primary radio button and for the Lun Name enter boot lun name (for example, Boot-Lun). Click OK to add the local lun image.



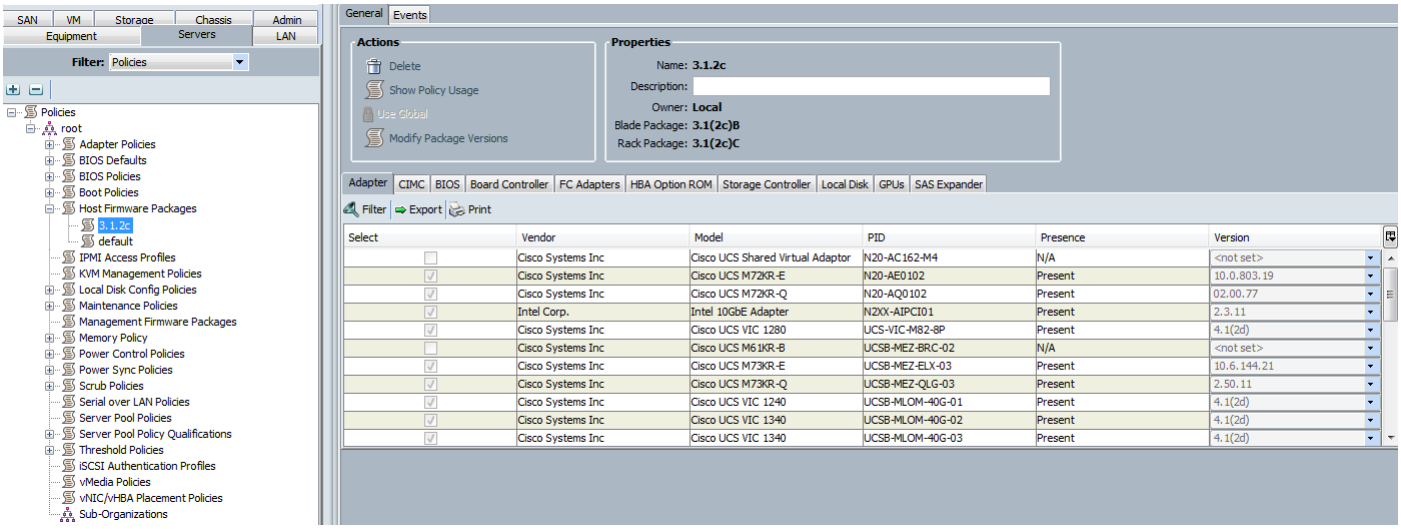
7. After the creation Boot Policy, you can view the created boot options as shown.



Creating Host Firmware Package Policy

Firmware management policies allow the administrator to select the corresponding packages for a given server configuration. These policies often include packages for adapter, BIOS, board controller, FC adapters, host bus adapter (HBA) option ROM, and storage controller properties. To create a firmware management policy for a given server configuration in the Cisco UCS environment, complete the following steps:

1. In Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Policies > root > Host Firmware Packages.
3. Right-click on Host Firmware Packages and select Create Host Firmware Package.
4. Enter the name of the host firmware package (for example, 3.1.2c).
5. Leave Simple selected.
6. Select the package versions for the different type of servers (Blade, Rack) in the deployment (for example, 3.1(2c) for Blade and Rack servers.
7. Click OK twice to create the host firmware package.



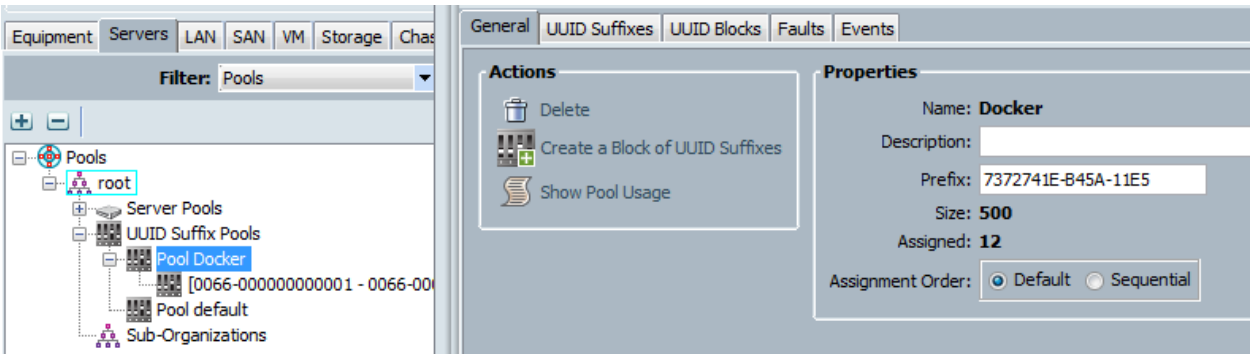
Creating Server Pools

Creating UUID Suffix Pool

To configure the necessary universally unique identifier (UUID) suffix pool for the Cisco UCS environment, complete the following steps:

1. From Cisco UCS Manager, select Servers tab in the navigation pane.
2. Select Servers > Pools > root > UUID Suffix Pools.
3. Right-click and select Create UUID Suffix Pool.
4. Specify a Name (for example, Docker) for the UUID suffix pool and click Next.
5. Click [+] Add to add a block of UUIDs. Alternatively, you can also modify the default pool and allocate/modify a UUID block.

6. Keep the From field at the default setting. Specify a block size (for example, 12) that is sufficient to support the available blade or server resources.
7. Click OK, click Finish and click OK again to create UUID Pool.

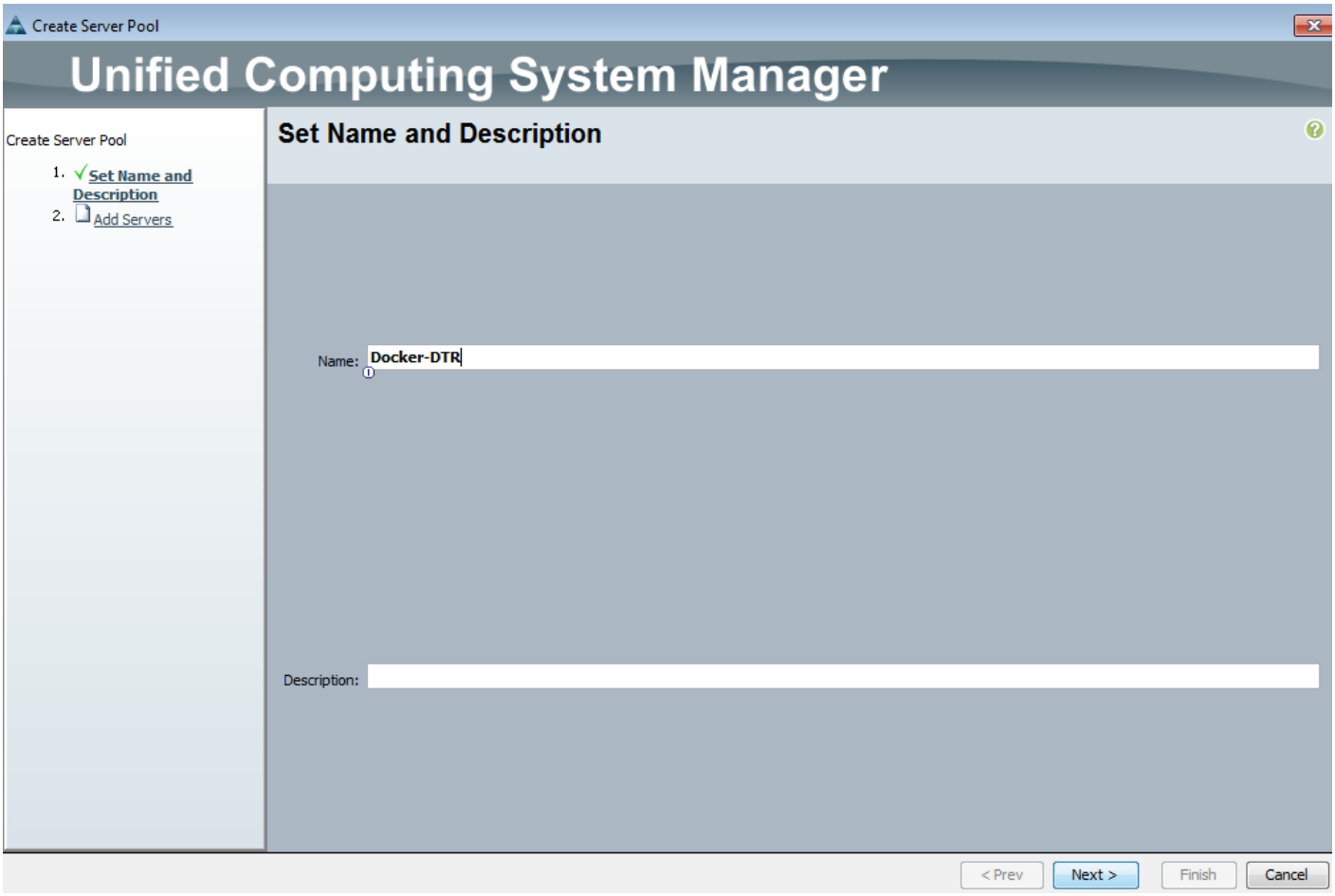


Creating Server Pools

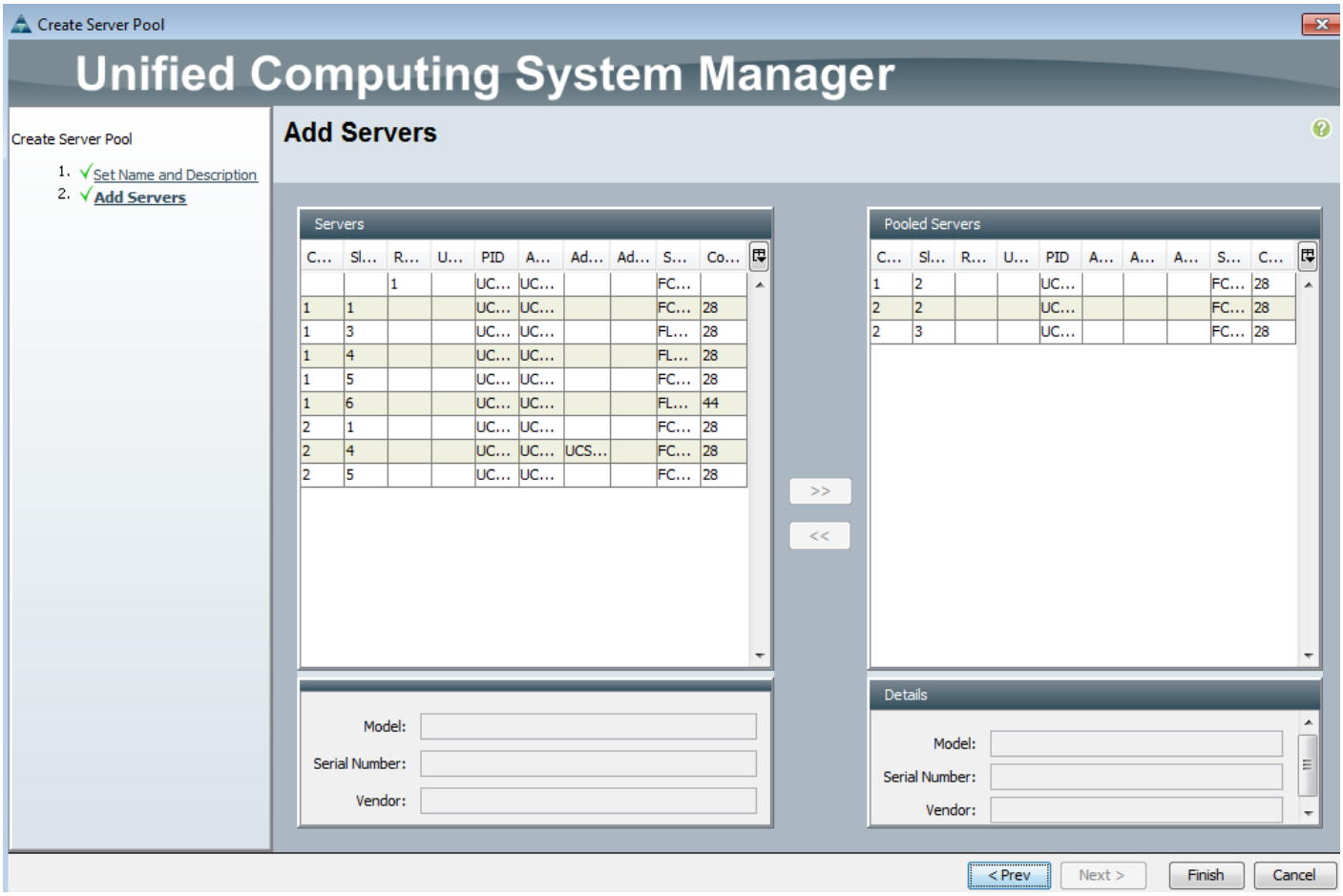
We have created three server pools, one each for DTR nodes, UCP controller nodes, and UCP nodes. Since we have three DTR nodes, three UCP-Ctrl nodes and 4 UCP nodes in our solution, we have created separate pools for each of these categories coming from two different chassis.

To configure the necessary server pool for the Cisco UCS environment, complete the following steps:

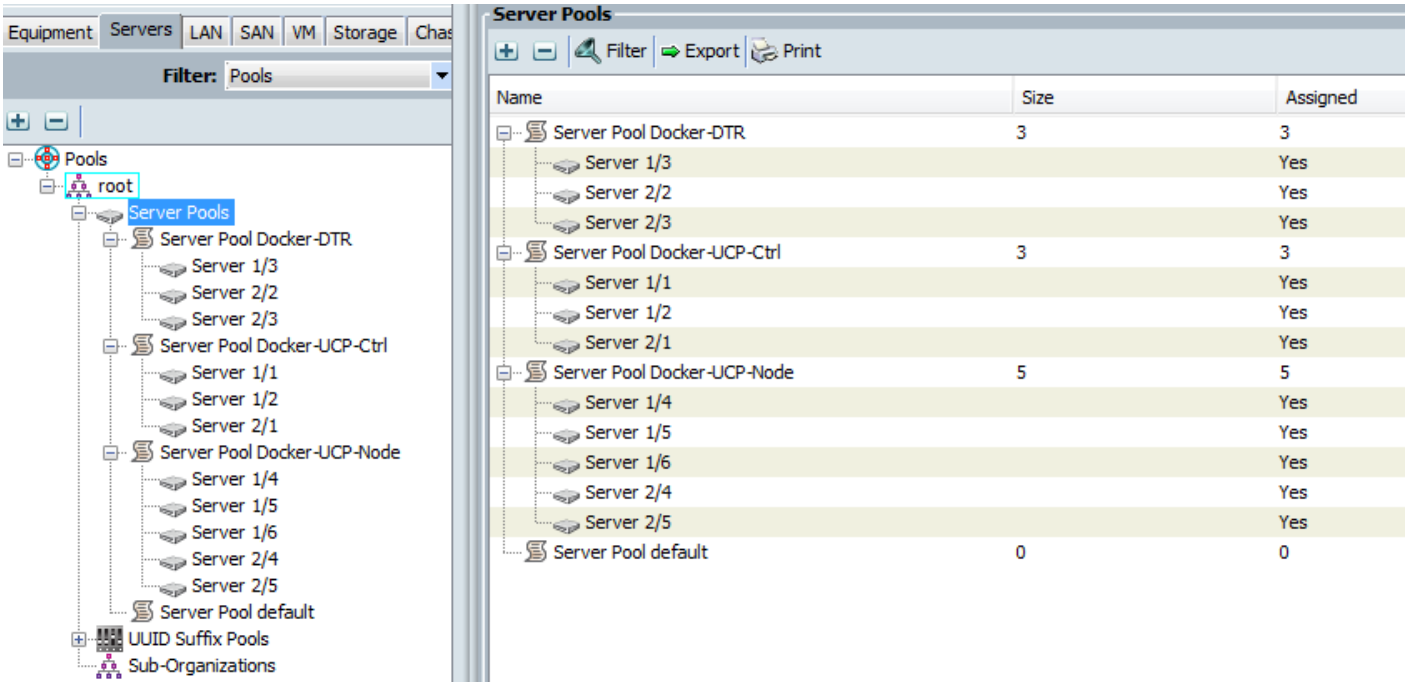
1. In Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Pools > root.
3. Right-click Server Pools and select Create Server Pool.
4. Enter name of the server pool (for example, Docker-DTR).
5. Optional: Enter a description for the server pool.
6. Click Next.



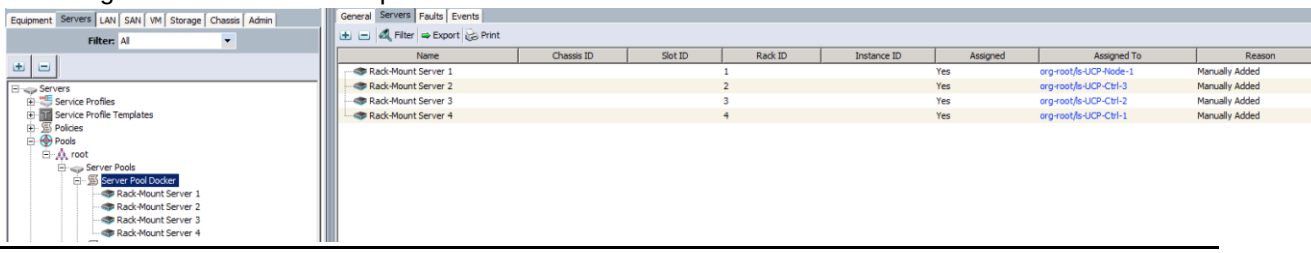
7. Select two (or more) servers to be used for the VMware management cluster and click >> to add them to the server pool.



8. Click Finish to complete.
9. Similarly create two more Server Pools (for example, Docker-Ctrl and Docker-UCP-Node). The created Server Pools can be viewed under Server Pools.



For second architecture one server pool (for example, Docker) is created using above steps and selecting rack servers into the pool:



Cisco UCS Configuration – Storage

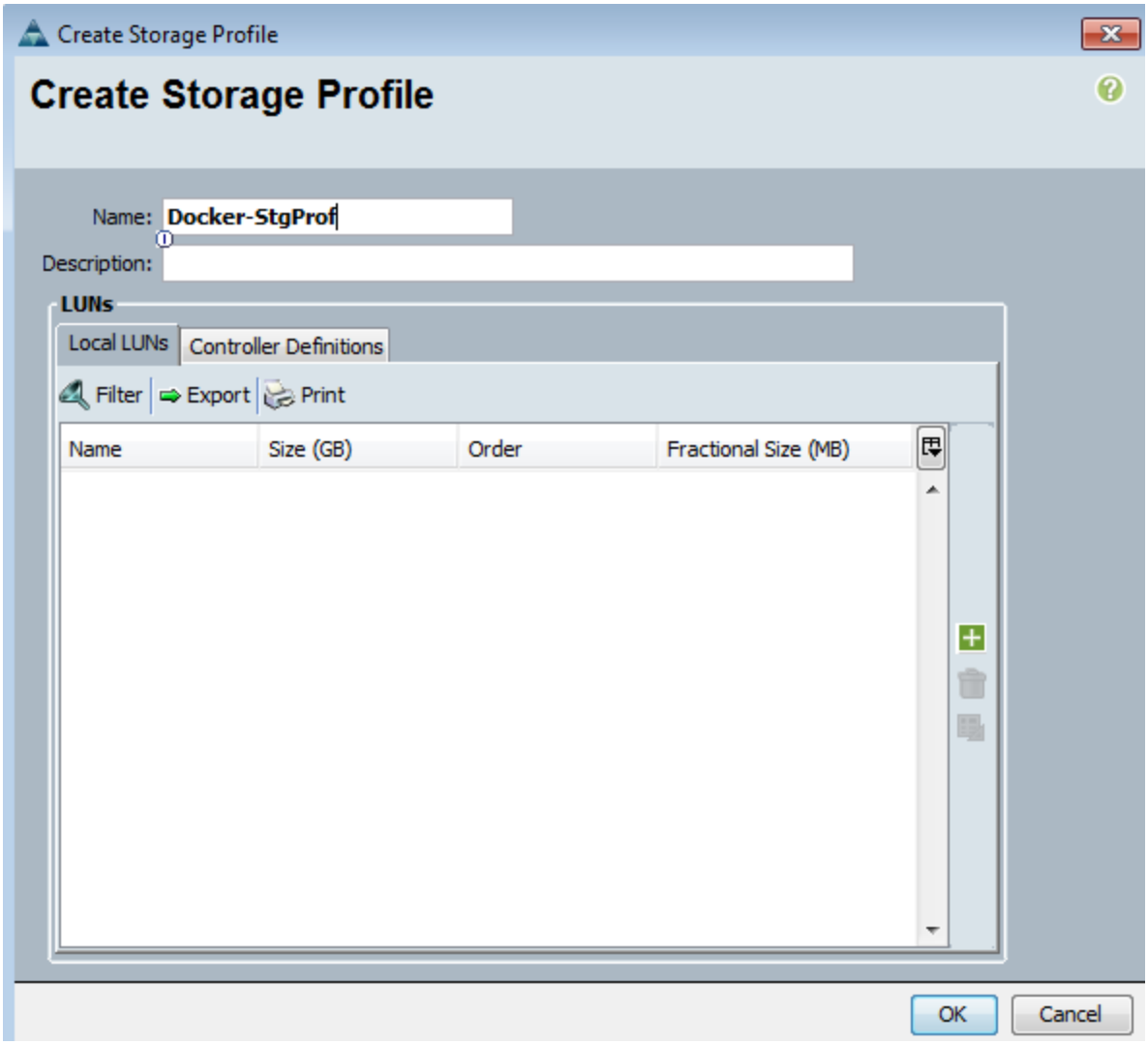
Creating Storage Profile

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.

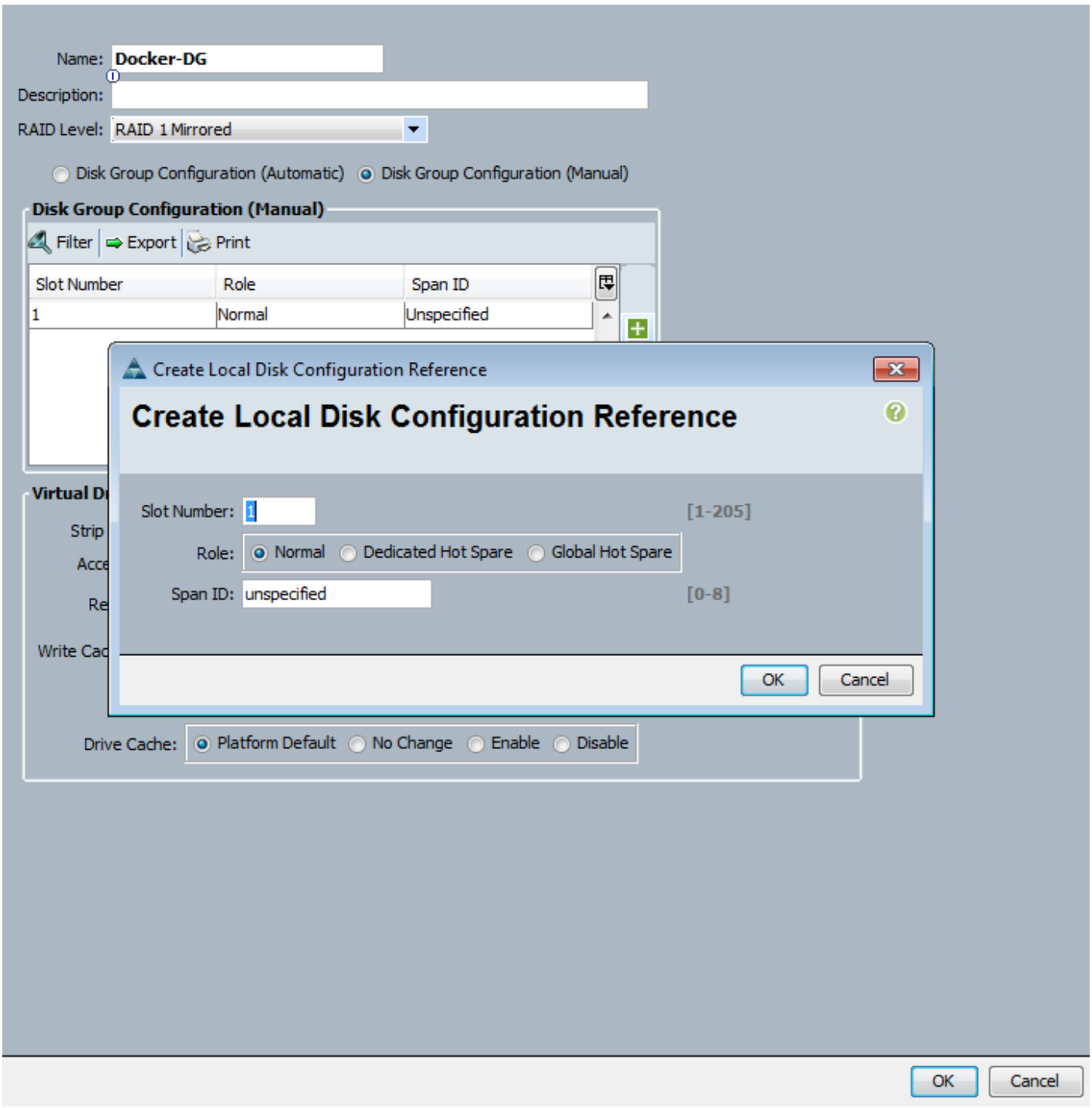
Having a storage profile created will reduce the task of configuring two virtual disks in the RAID Controller Option ROM or create a custom file system layout at the time of OS installation.

We have created a Storage profile with two local luns one each for boot and data. Complete the following to create a storage profile:

1. In Cisco UCS Manager, click Storage tab in the navigation pane.
2. Select Storage > Storage Profiles.
3. Right-click Storage Profiles and select Create Storage Profile.
4. Enter the name for the Storage Profile (for example, Docker-StgProf).

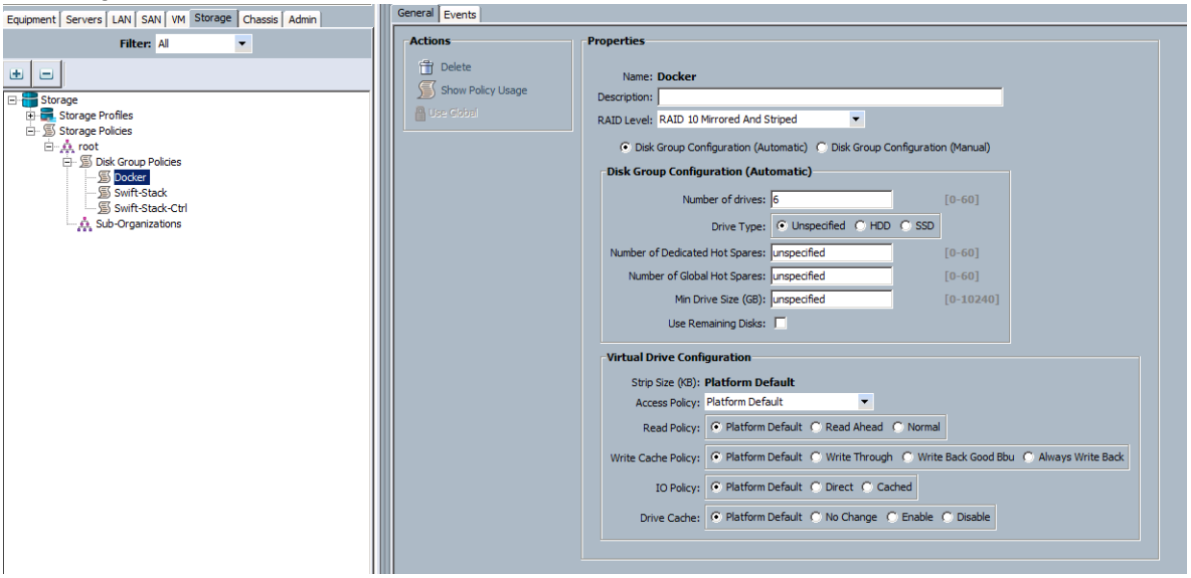


5. In the Local Luns tab, click + on the right plane of the Create Storage Profile Window.
6. Create Local Lun window appears. Keep the Create Local Lun radio button selected.
7. Enter the lun name (for example, Boot-Lun) and specify the desired size (for example, 60GB).
8. For select Disk Group configuration, Click Create Disk Group Policy.
9. Enter the Disk Group name (for example, Docker-DG). Keep the RAID Level as RAID 1 Mirrored, since blades come with only 2 disks.
10. Select Disk Group Configuration (Manual) radio button. Click + to add two slots; one for Boot-Lun and the other for Data-Lun. Keep the other fields as is and click OK.



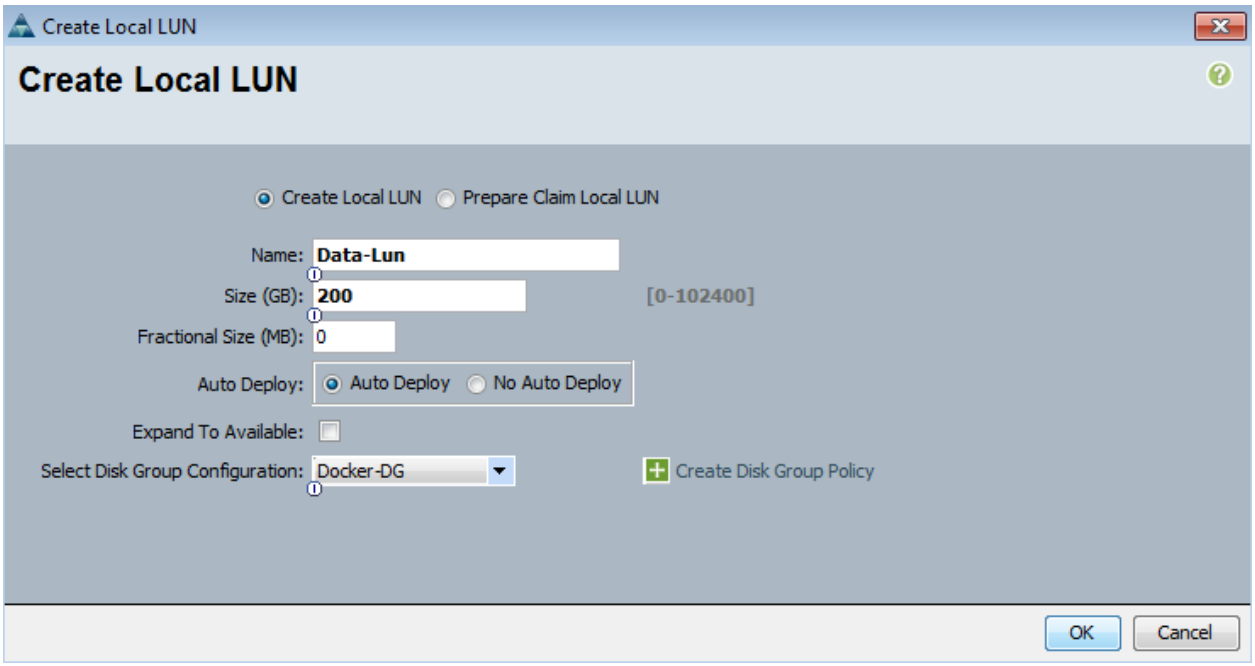
11. Repeat step 10 to create another Local Disk Configuration Reference. Click OK.

For second architecture Disk Group Policy uses RAID-10 with Automatic configuration option by selecting all 6 internal disks:

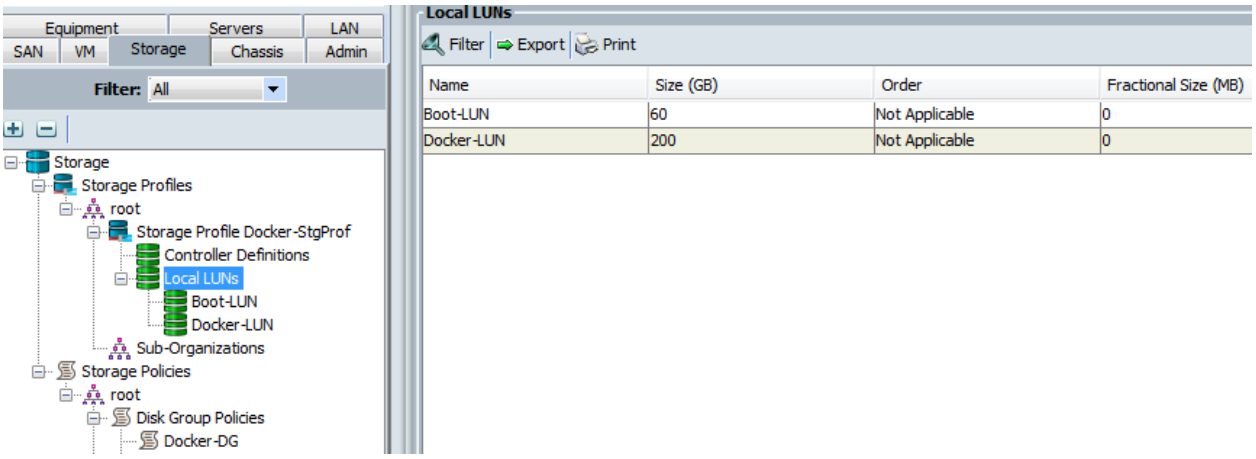


RAID-10 provides mirrored and stripped pair of disks thereby giving redundancy and performance. A RAID-10 requires a minimum of 4 disks, we have 6 disks on each of the C-Series servers. This give us around 3TB of storage space for Docker run-time and local image store and greater storage scalability.

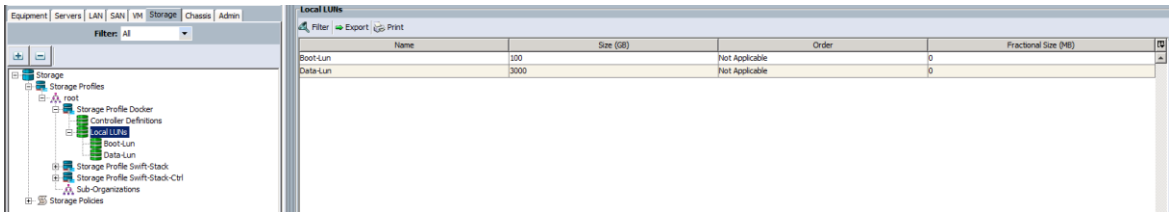
12. Select the created disk group from the Select Disk Group Configuration drop-down (for example, Docker-DG). Click OK to create Boot-Lun.
13. Repeat step 7 to create Data-Lun. Enter the appropriate name (for example, Data-Lun) and size (for example, 200GB) and select Disk Group Configuration (for example, Docker-DG). Click OK to create Data-Lun.



14. The created Local Luns can be view under Storage Profiles and Disk Group Policy under Storage Policies.



For second environment with C-Series rack servers, Boot-LUN and Data-LUN is created with 100GB and 3000GB size:



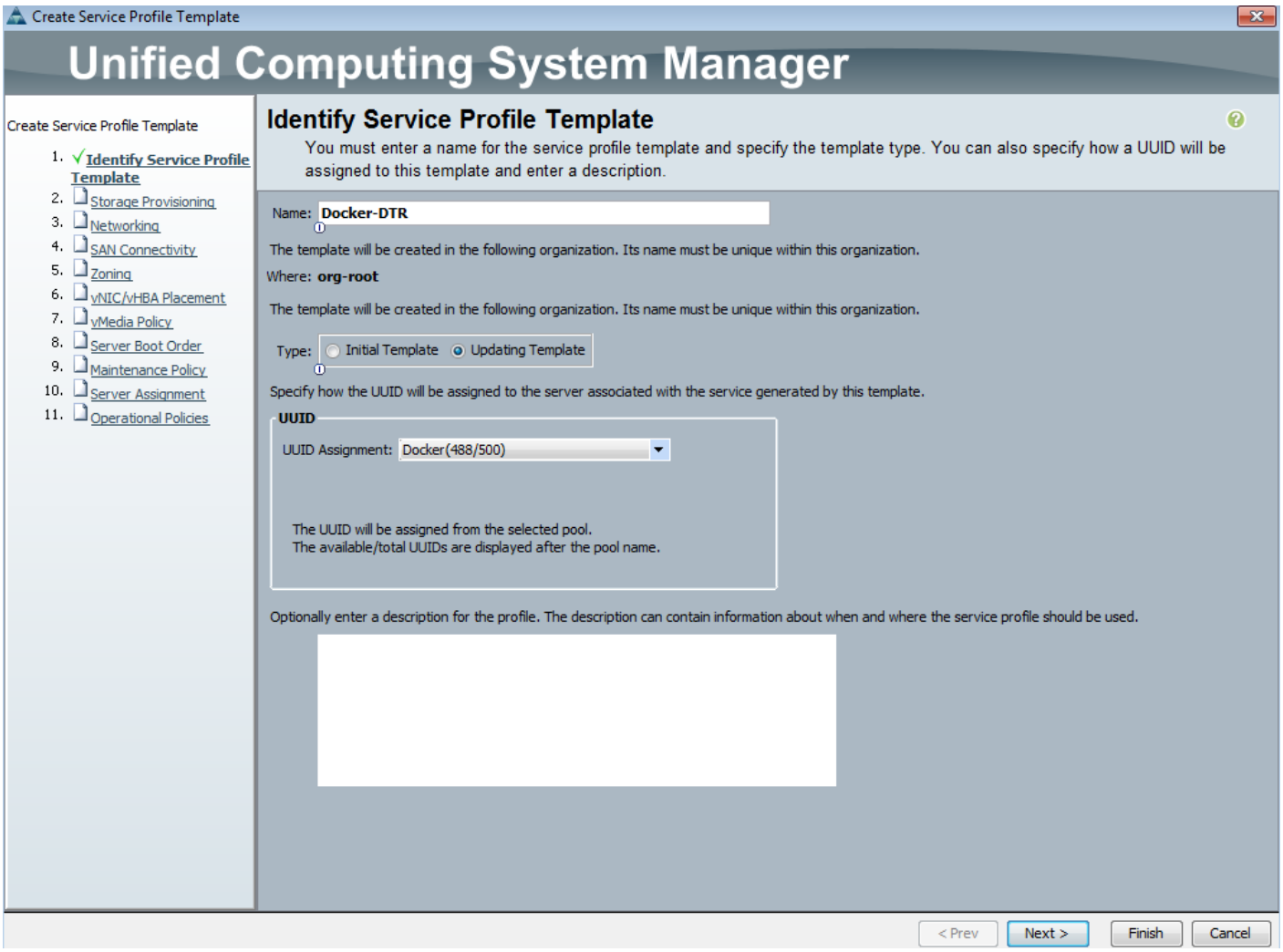
Creating Service Profile Templates

In this procedure, three service profile templates are created: one each for DTR nodes, UCP controller nodes and UCP nodes. The first profile template is created, then cloned and renamed for the second and third profiles. Since there are three DTR, three UCP controller and four UCP nodes, we instantiate service profiles for these categories from the three different service profile templates.

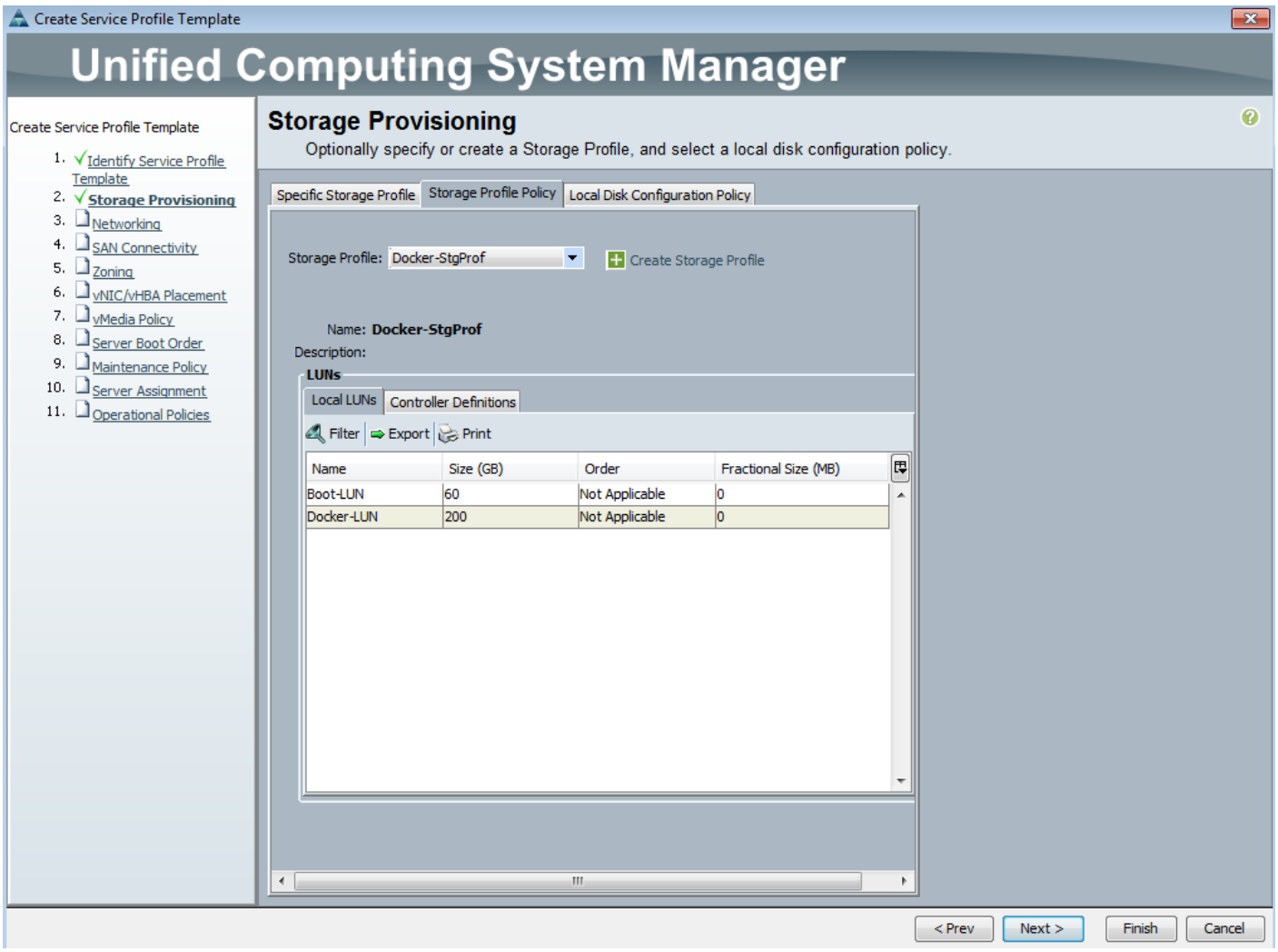
Creating Service Profile Template

To create service profile templates (for example, DTR nodes), complete the following steps:

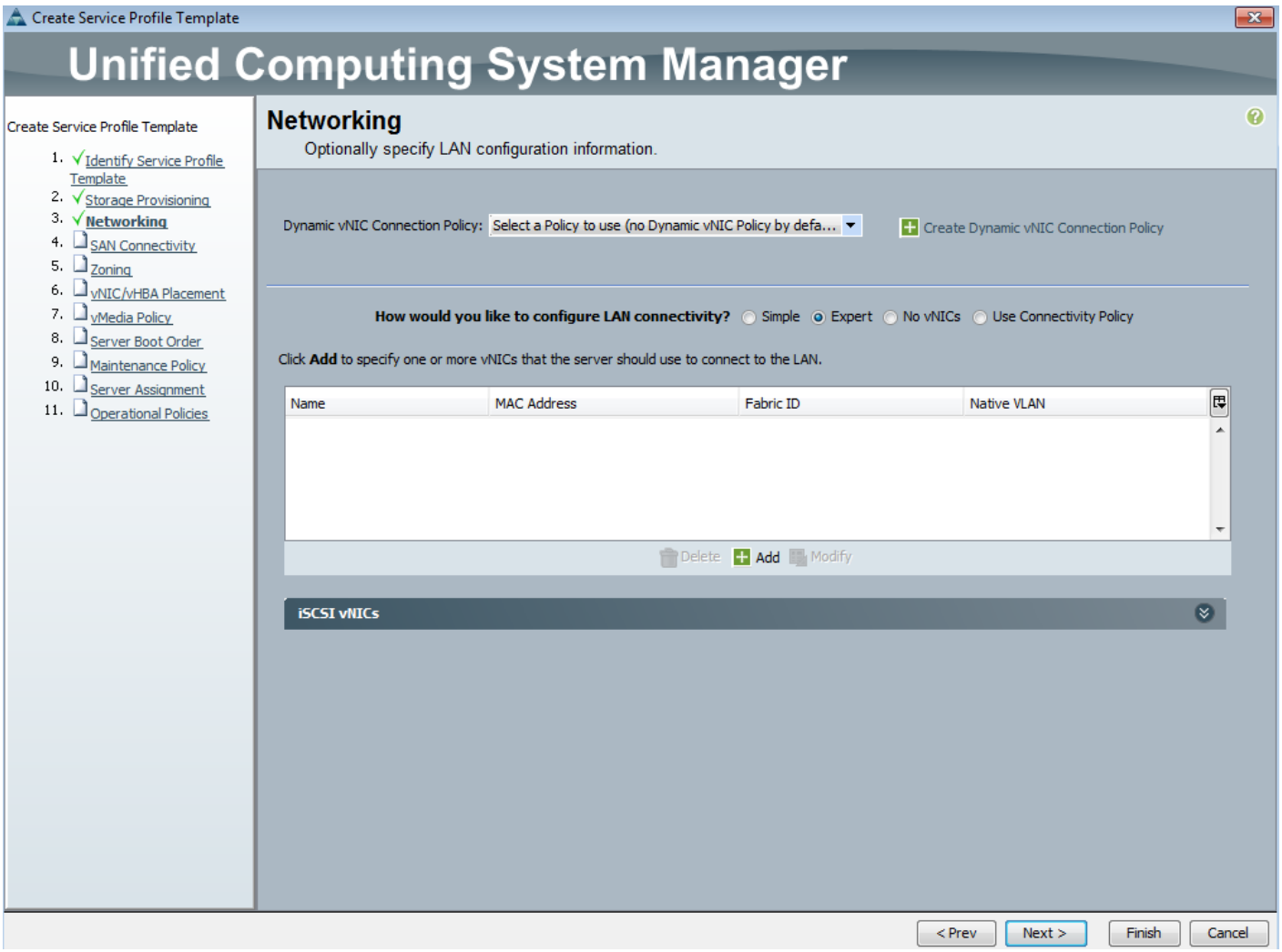
1. From Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Servers > Service Profile Template > root.
3. Right-click root and select Create Service Profile Template to open the Create Service Profile Template wizard.
4. In the Identify the Service Profile Template screen, configure the following:
   a. Enter name (for example, Docker-DTR) for the service profile template.
   b. Select Updating Template radio button.
   c. Under UUID, select the previously configured UUID pool (for example, Docker).
   d. Click Next.



5. In the Storage Provisioning screen, configure the following:
   a. Go to Storage Profile Policy tab.
   b. In the Storage profile drop-down, select a policy. Choose the previously configured policy (for example, Docker-StgProf). Local Luns tab lists the previously configured local luns.
   c. Click Next.



6. In the Networking screen, configure the following:
   a. Restore the default setting for Dynamic vNIC Connection Policy.
   b. Click Expert radio button to configure the LAN connectivity.



c. Click on [+] Add to add a vNIC to the template.

d. In the Create vNIC dialog box:

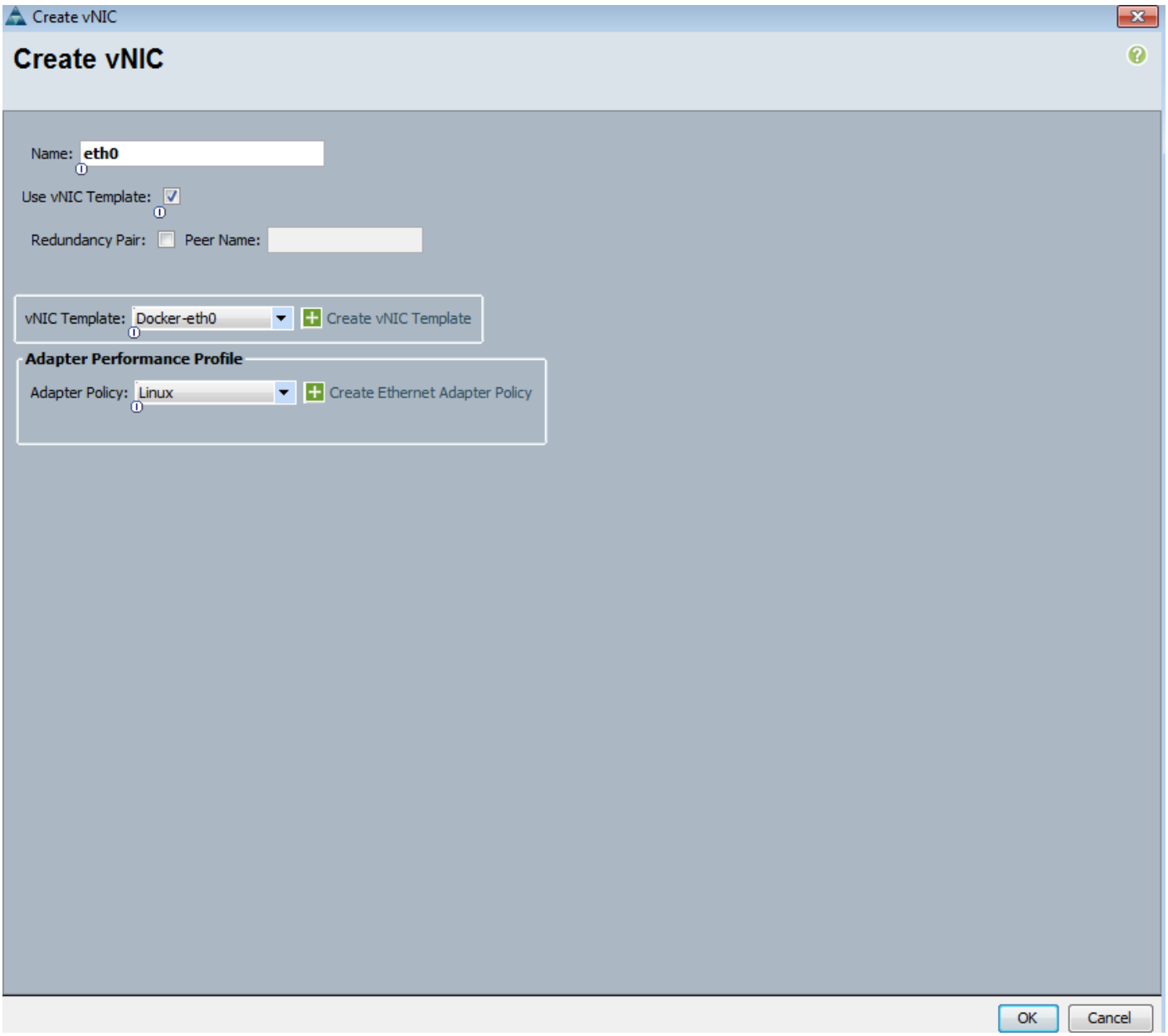
   – Enter the name (for example, eth0) of the vNIC.

   – Check the Use vNIC Template check box.

   – In the vNIC Template list, choose the previously created vNIC Template for Fabric A boot (for example, Docker-eth0).

   – In the Adapter Policy list, choose Linux.

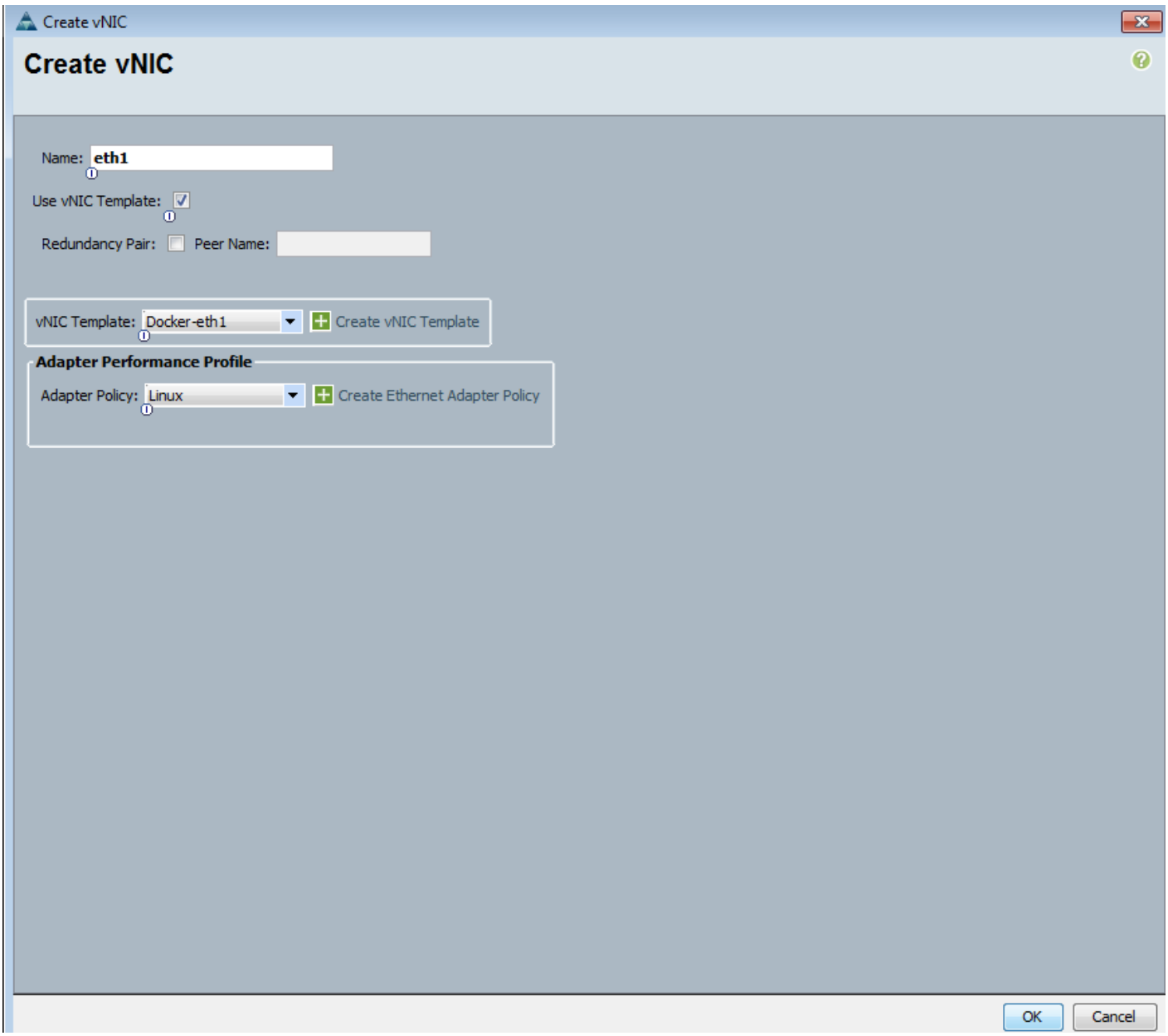
   – Click OK to add this vNIC to the template.



e. Click on [+] Add to add a 2nd vNIC to the template.

f. In the Create vNIC dialog box:

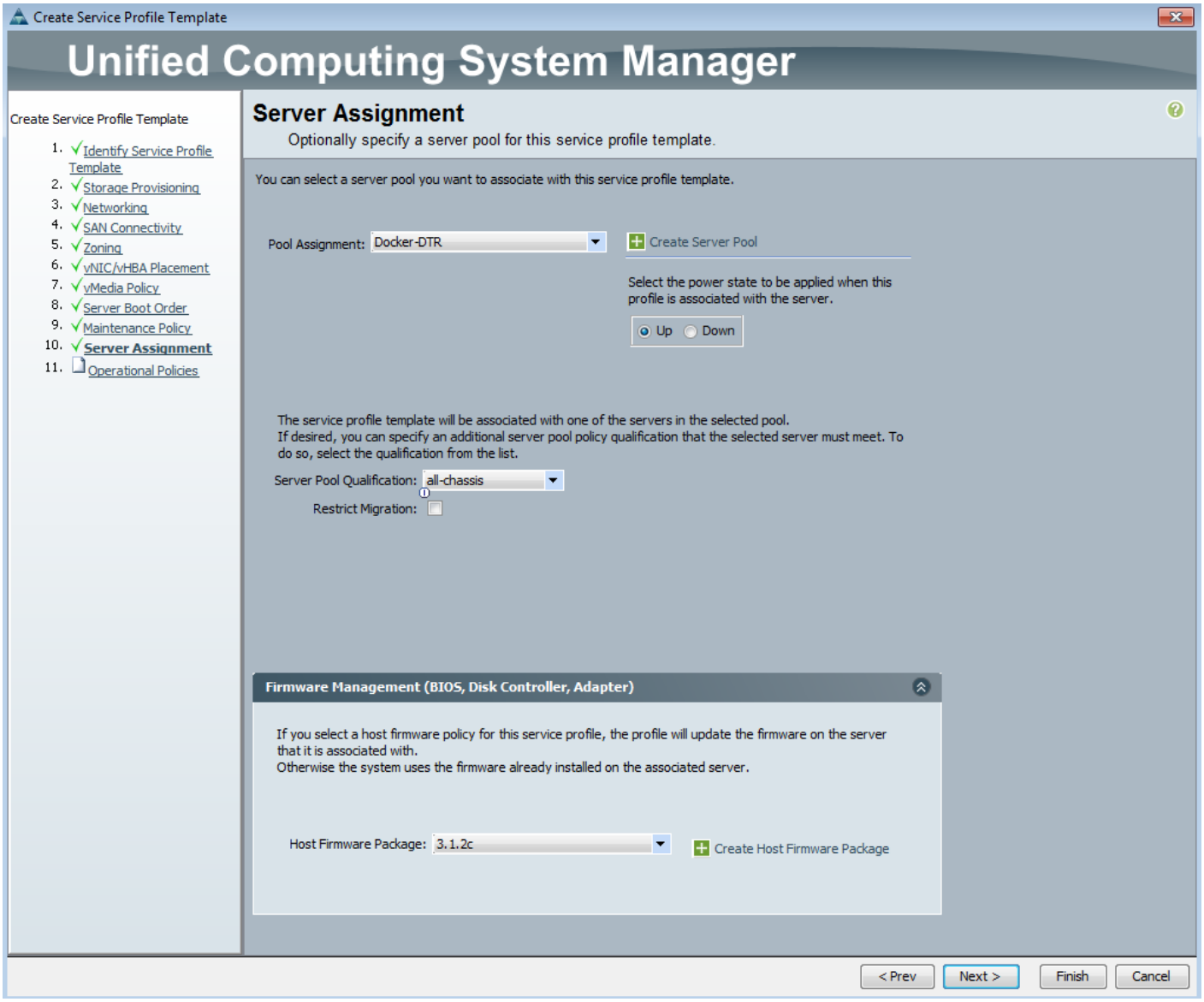
- Enter the name (for example, eth1) of the vNIC.
- Check the Use vNIC Template check box.
- In the vNIC Template list, choose the previously created vNIC Template for Fabric B boot (for example, Docker-eth1).
- In the Adapter Policy list, choose Linux.
- Click OK to add this vNIC to the template.



g. Review the configuration on the Networking screen of the wizard. Make sure that both the vNICs were created. Click Next.

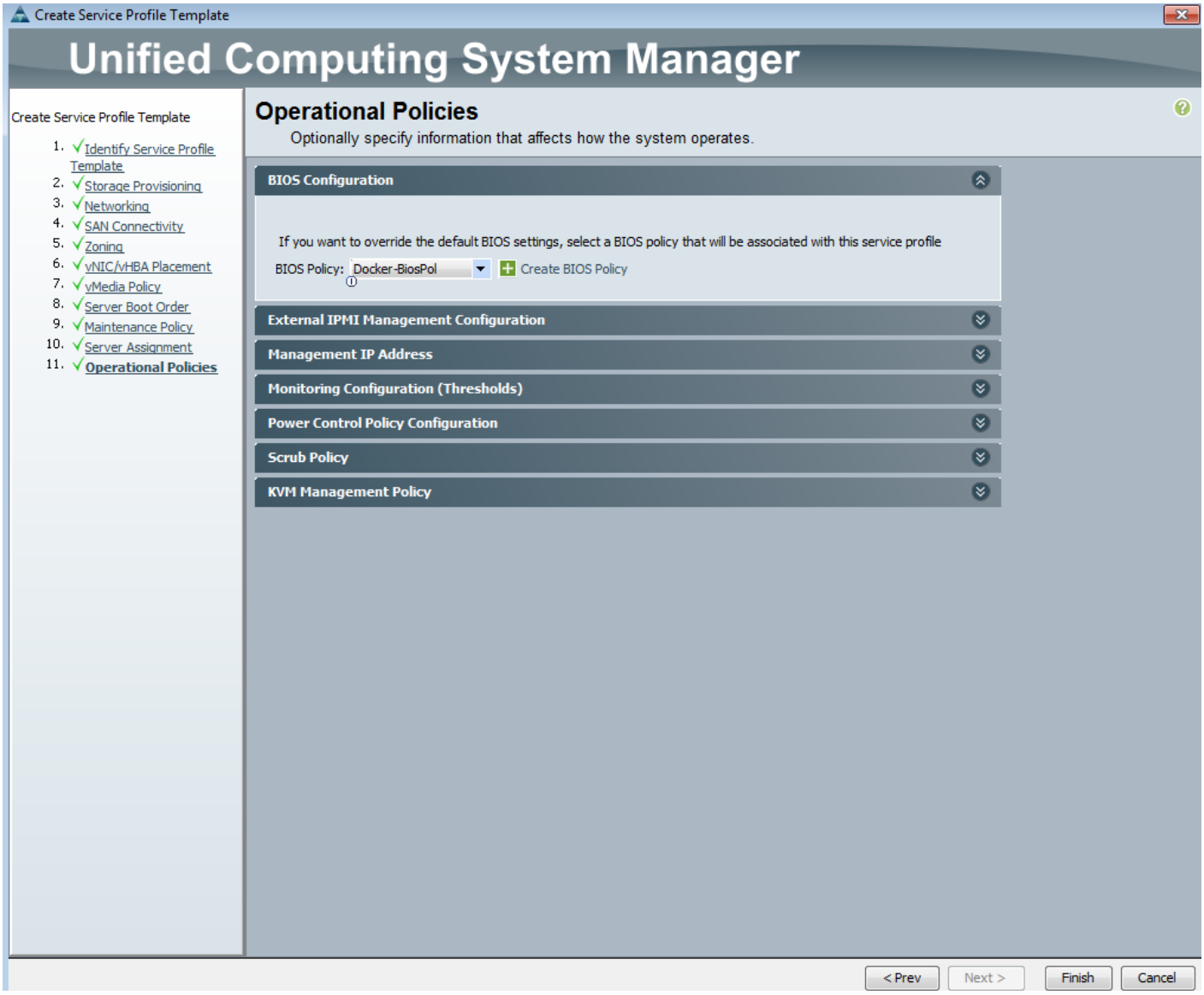
7. Click Next in the SAN Connectivity, Zoning, vNIC/vHBA Placement, and vMedia Policy screens.
8. In the Set Boot Order screen, select the previously created boot policy from the Boot Policy drop-down (for example, Docker-LocalBoot).

9. Click Next.
10. Click Next in Maintenance Policy screen.
11. In the Server Assignment screen, configure the following:
    a. For Pool Assignment, choose the previously created policy from the list (for example, Docker-DTR).
    b. Leave the Power State as UP for when the Profile is applied to a server
    c. For Server Pool Qualification, select the previously created policy from the list (for example, all-chassis).
    d. Expand the Firmware Management section. For the Host Firmware Package, select the previously selected policy from the list (for example, 3.1.2c).
    e. Click Next.

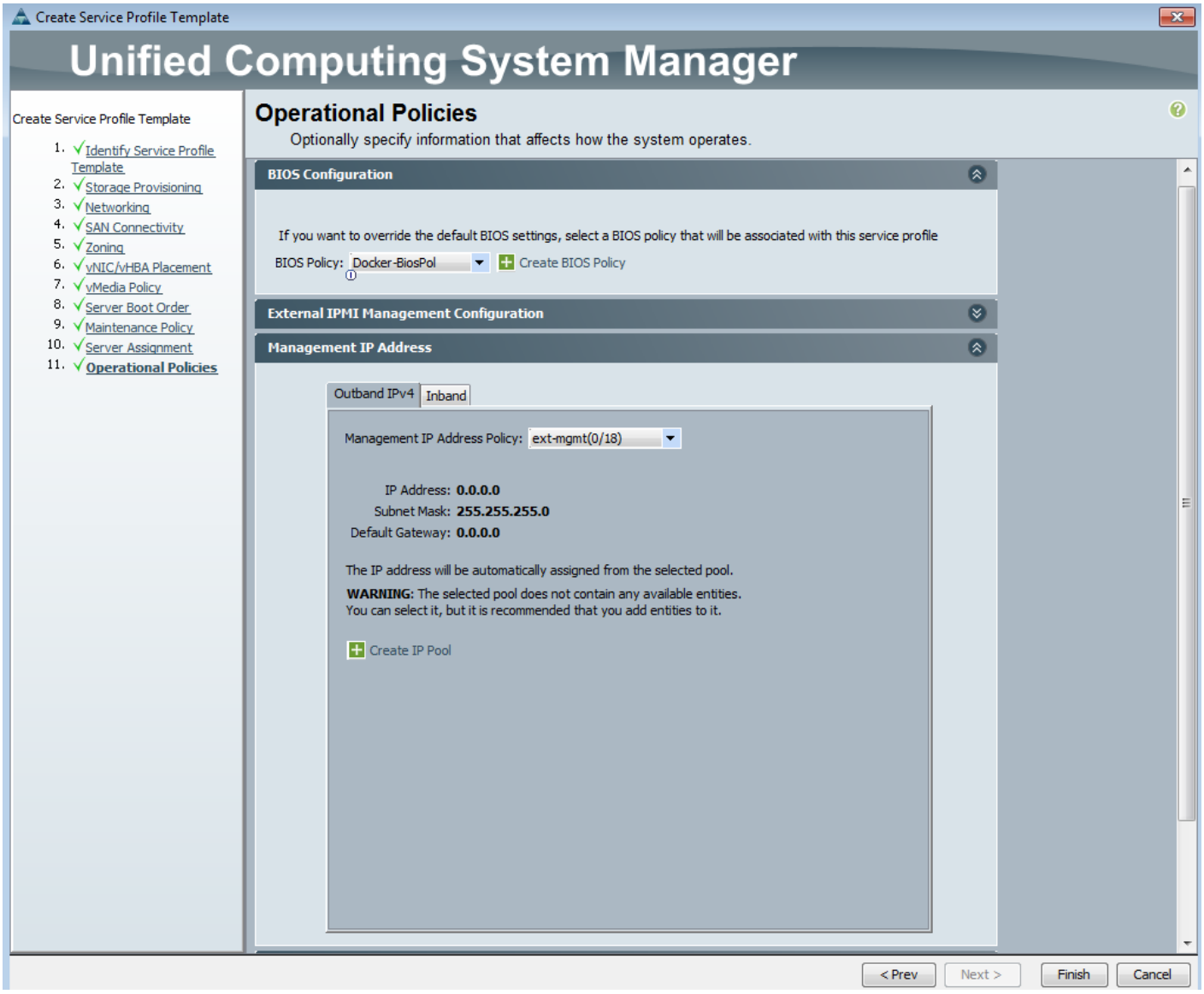


12. In the Operation Policies screen, configure the following:

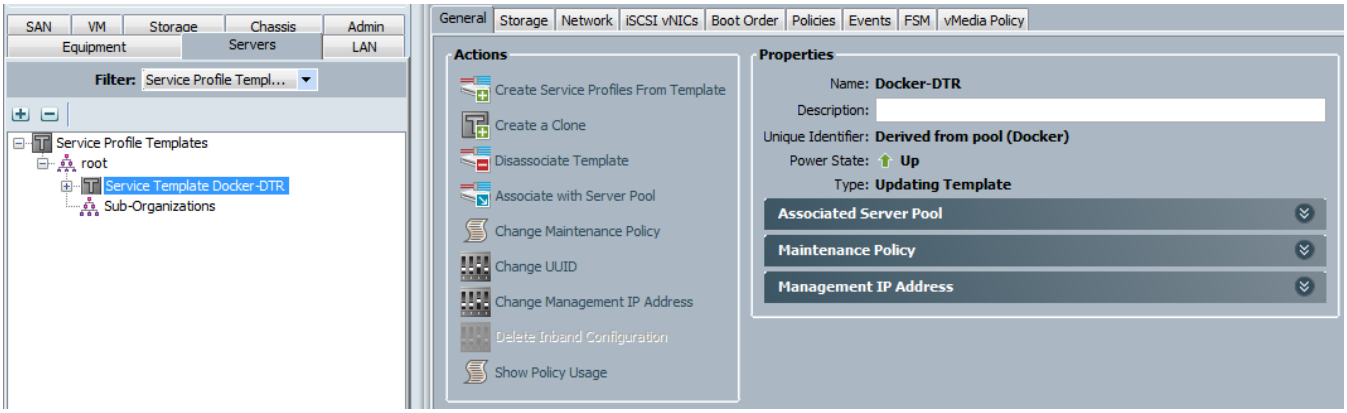
    a. For the BIOS Policy list, select the previously configured policy (for example, Docker-BiosPol).



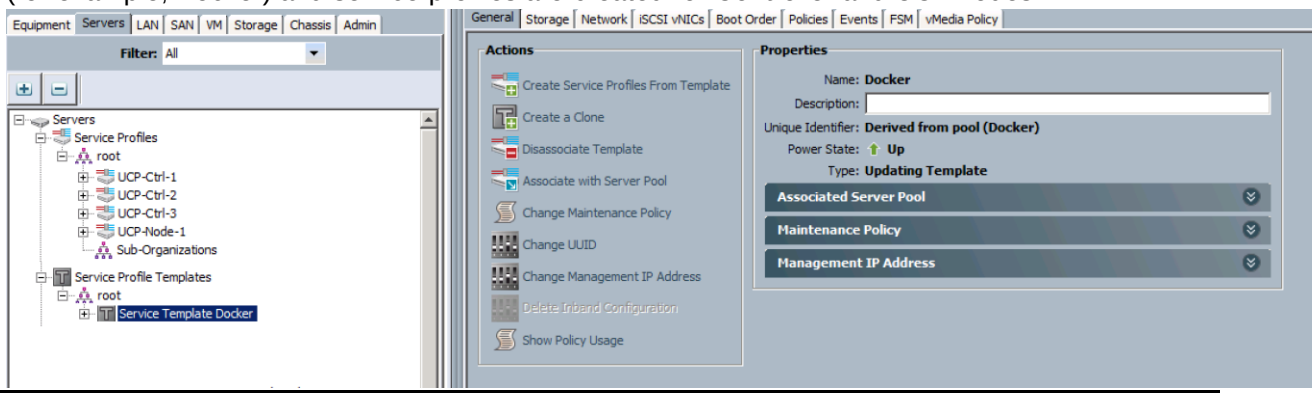
13. Expand Management IP Address, select the IP address pool (for example, ext-mgmt(0/18)) from the management IP Address policy drown-down.



14. Click Finish to complete the creation of the Service Profile Template. Created service profile template will get listed under Service Profile Templates as shown in the below figure.



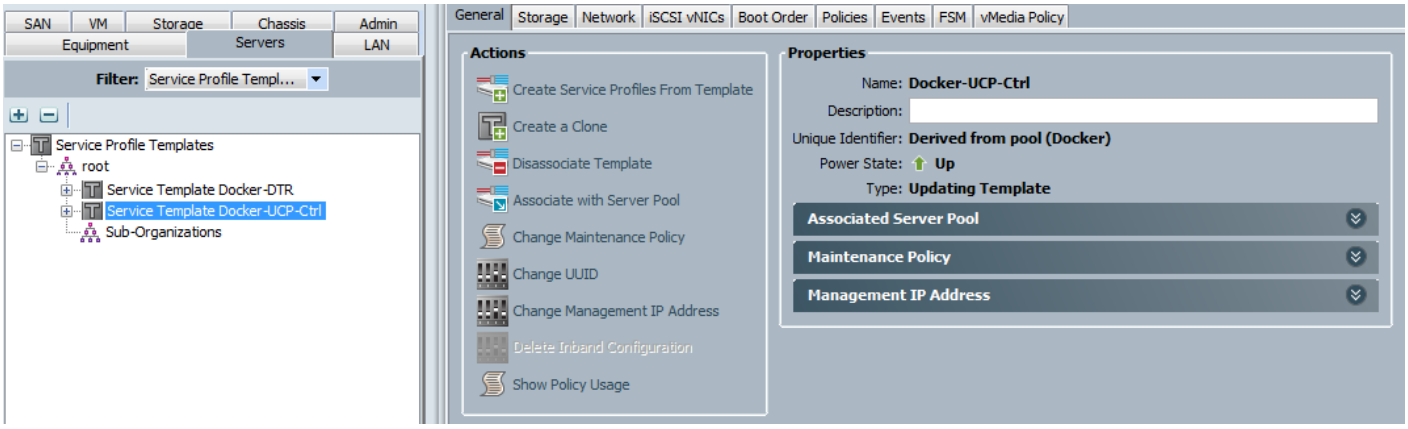
For second architecture on C-series rack servers, we create one common service profile template (for example, Docker) and service profiles are created for Controller and UCP nodes:



Creating Service Profile Template for UCP Controller Nodes

Repeat the steps 1 to 14 detailed in the previous sub-section for creating a service profile template for UCP controller nodes.

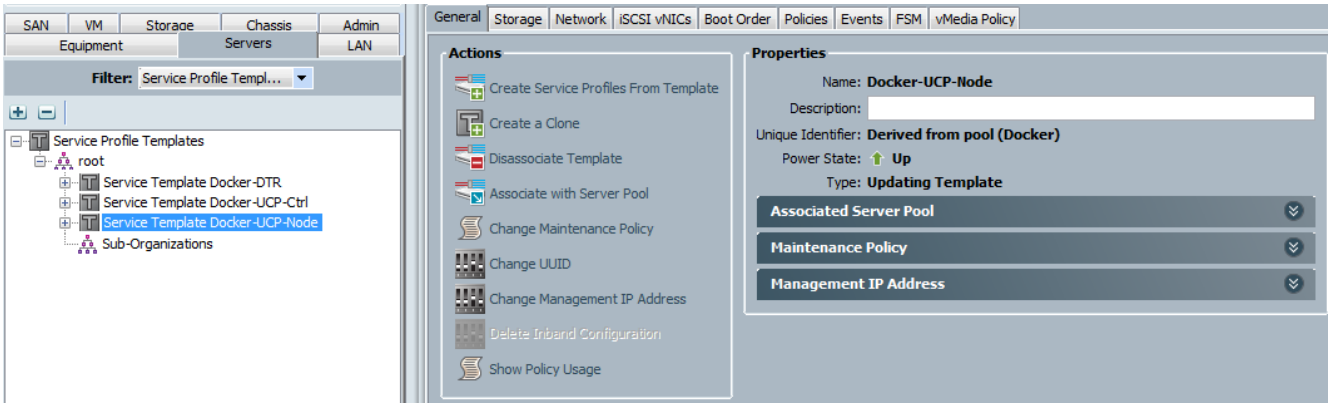
After creating the service profile template, the template for UCP controller nodes will look similar to the below screenshot.



Creating Service Profile Template for UCP Nodes

Repeat the steps 1 to 14 detailed in the previous sub-section for creating a service profile template for UCP nodes.

After creating the service profile template, the template for UCP nodes will look similar to the below screenshot.



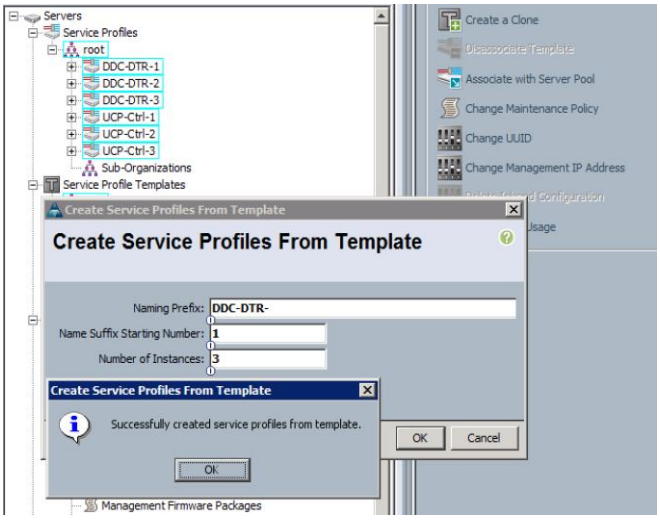
Service Profile Instantiation and Association

Service Profile Instantiation

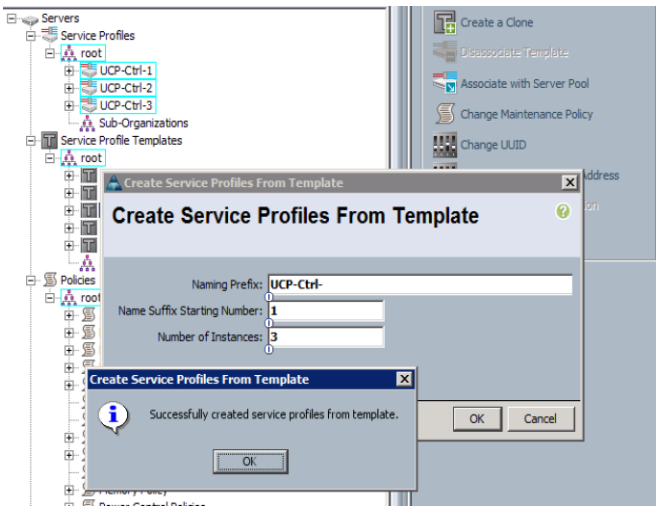
In this process we have instantiated service profiles for DTR, UCP controller and UCP nodes from their respective templates that we created in the previous section.

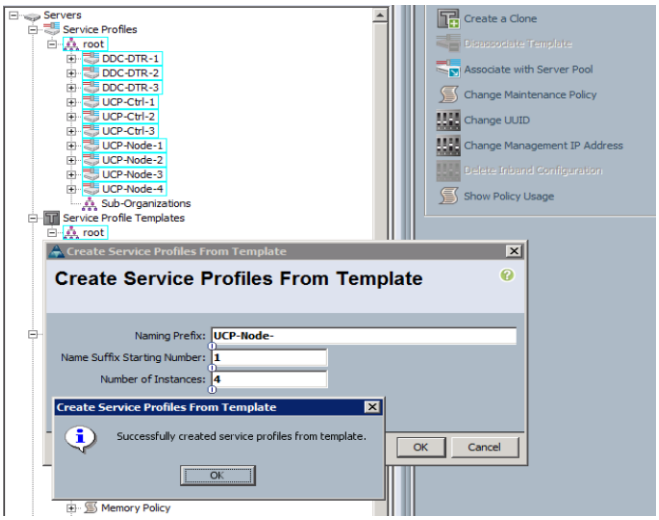
To create service profiles from template, complete the following steps:

1. From Cisco UCS Manager, click Servers tab in the navigation pane.
2. Expand Servers > Service Profile Templates.
3. Right-click on the specific template (for example, Docker-DTR) and select Create Service Profiles from Template to open the Create Service Profile window.
4. In the Create Service Profile window, enter the profile name (for example, DDC-DTR-), enter the suffix to start the instances and enter the number of instances to be instantiated.



5. Similarly instantiate other two service profiles (for example, Docker-UCP-Ctrl and Docker-UCP-Nodes.



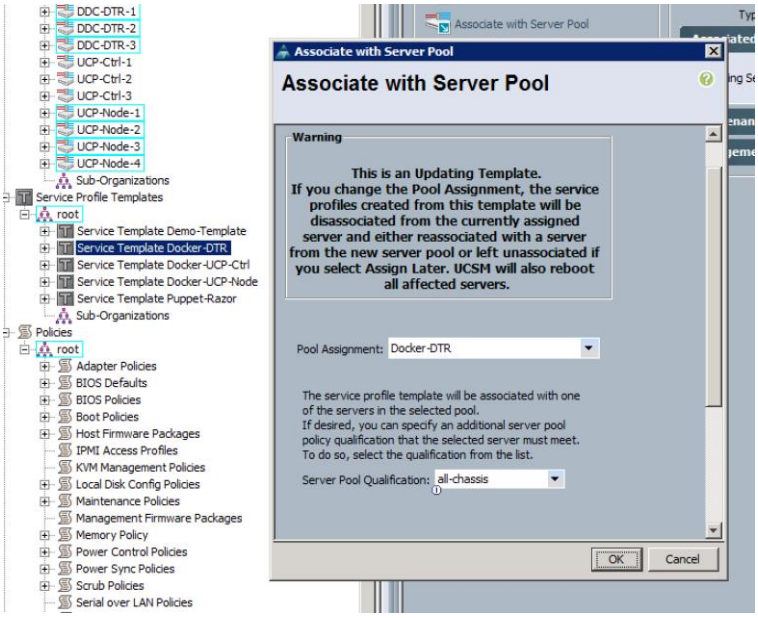


Associating Service Profile Templates to Server Pools

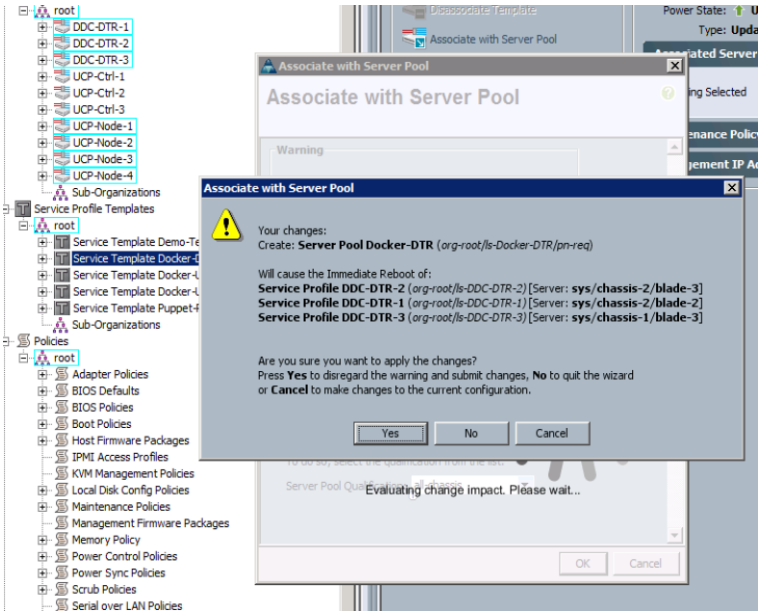
In this process we associate the three service profile templates to the previously created server pools for DTR, UCP controller and UCP nodes.

Follow the steps to complete server pool association:

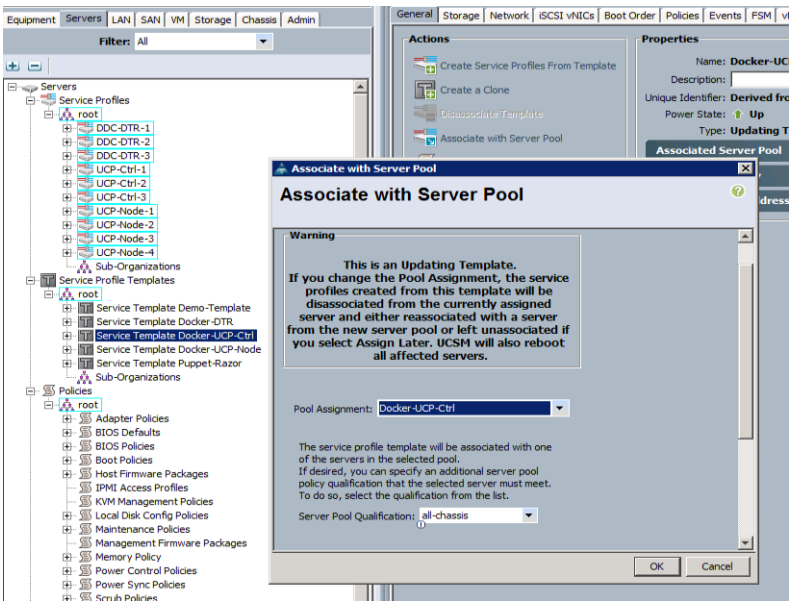
1. From Cisco UCS Manager, click Servers tab in the navigation pane.
2. Expand Servers > Service Profile Templates.
3. Right-click on the specific template (for example, Docker-DTR) and select Associate with Server Pools to open the Associate with Server Pool window.
4. Select the specific sever pool that you want to associate the template to (for example, Docker-DTR).
5. Right-click on the specific template (for example, Docker-DTR) and select Associate with Server Pools to open the Associate with Server Pool window. Select the Server Pool Qualification as all-chassis as we have the DTR, UCP controller and UCP nodes distributed in two different chassis.

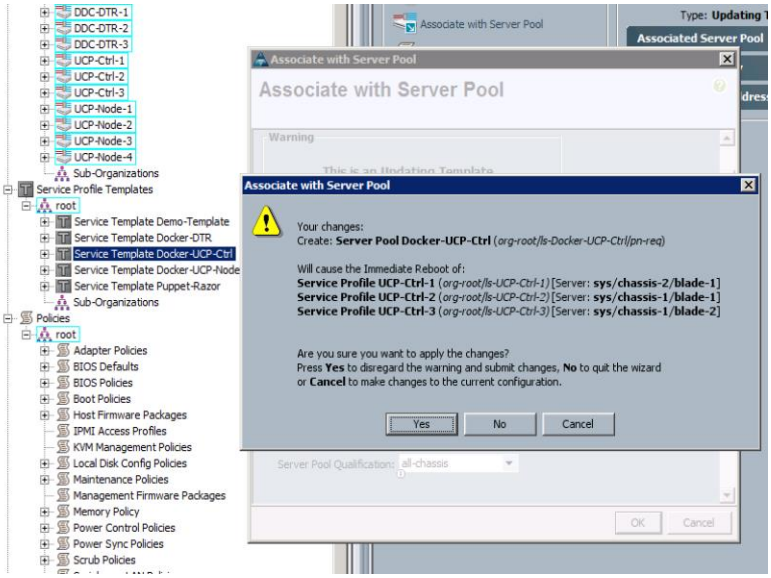


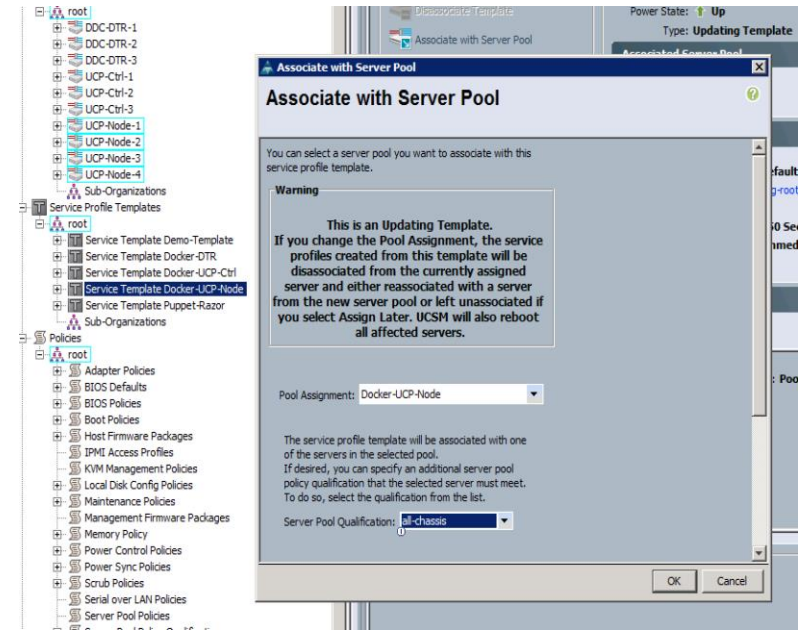
6. Click OK. A pop-up window will open prompting you to confirm the server pool association.

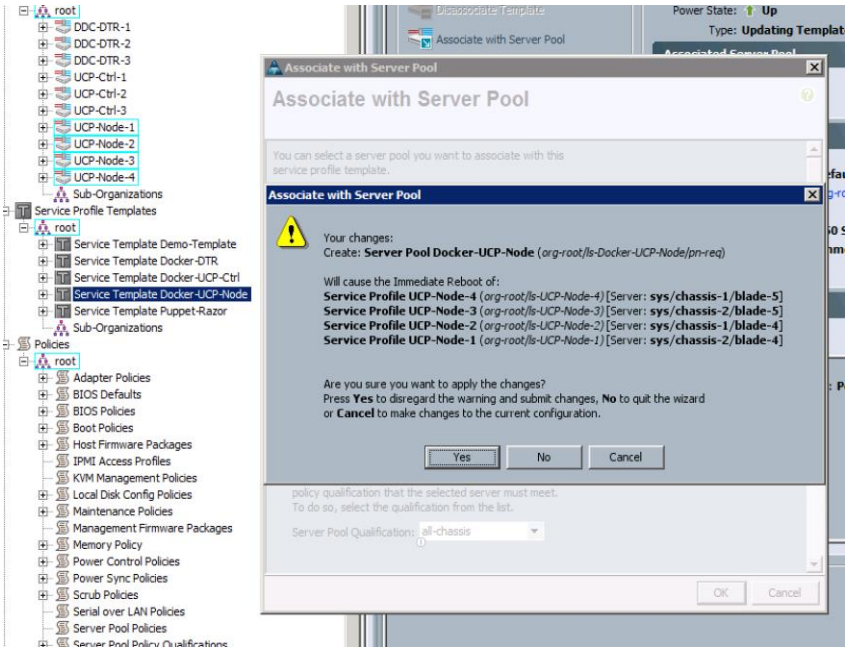


7. Similarly associate the other two templates to the specific server pools (for example, Docker-UCP-Ctrl and Docker-UCP-Node) as shown below.

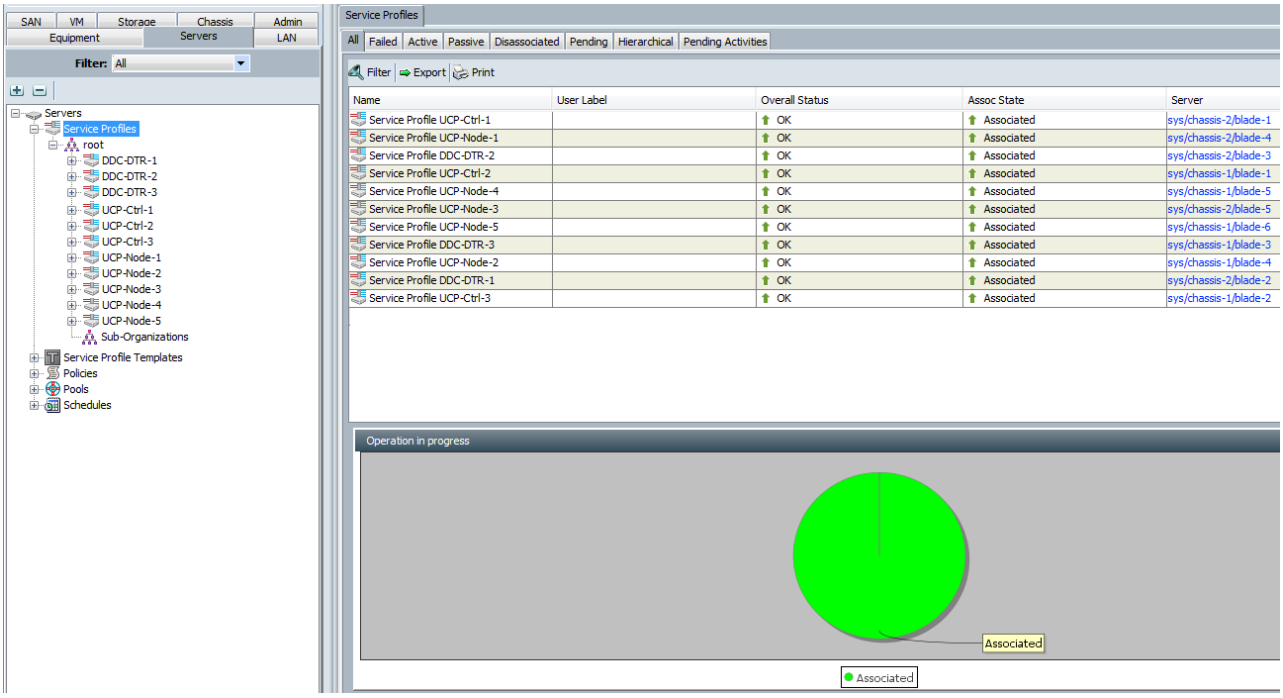

Equipment
Servers
LAN
SAN
VM
Storage
Chassis
Admin
Filter: All
Servers
Service Profiles
root
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
Sub-Organizations
Service Profile Templates
root
Service Template Demo-Template
Service Template Docker-DTR
Service Template Docker-UCP-Ctrl
Service Template Docker-UCP-Node
Service Template Puppet-Razor
Sub-Organizations
Policies
root
Adapter Policies
BIOS Defaults
BIOS Policies
Boot Policies
Host Firmware Packages
IPMI Access Profiles
KVM Management Policies
Local Disk Config Policies
Maintenance Policies
Management Firmware Packages
Memory Policy
Power Control Policies
Power Sync Policies
Scrub Policies
General
Storage
Network
iSCSI vNICs
Boot Order
Policies
Events
FSM
Actions
Create Service Profiles From Template
Create a Clone
Associate with Server Pool
Properties
Name: Docker-UC
Description:
Unique Identifier: Derived fre
Power State: Up
Type: Updating T
Associated Server Pool
Associate with Server Pool
Associate with Server Pool
Warning
This is an Updating Template.
If you change the Pool Assignment, the service profiles created from this template will be disassociated from the currently assigned server and either reassociated with a server from the new server pool or left unassociated if you select Assign Later. UCSM will also reboot all affected servers.
Pool Assignment: Docker-UCP-Ctrl
The service profile template will be associated with one of the servers in the selected pool.
If desired, you can specify an additional server pool policy qualification that the selected server must meet.
To do so, select the qualification from the list.
Server Pool Qualification: all-chassis
OK
Cancel


DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
Sub-Organizations
Service Profile Templates
root
Service Template Demo-Template
Service Template Docker-DTR
Service Template Docker-UCP-Ctrl
Service Template Docker-UCP-Node
Service Template Puppet-Razor
Sub-Organizations
Policies
root
Adapter Policies
BIOS Defaults
BIOS Policies
Boot Policies
Host Firmware Packages
IPMI Access Profiles
KVM Management Policies
Local Disk Config Policies
Maintenance Policies
Management Firmware Packages
Memory Policy
Power Control Policies
Power Sync Policies
Scrub Policies
Associate with Server Pool
Type: Updating T
Associated Server Pool
Associate with Server Pool
Associate with Server Pool
Warning
Associate with Server Pool
Your changes:
Create: Server Pool Docker-UCP-Ctrl (org-root/ls-Docker-UCP-Ctrl/pn-req)
Will cause the Immediate Reboot of:
Service Profile UCP-Ctrl-1 (org-root/ls-UCP-Ctrl-1) [Server: sys/chassis-2/blade-1]
Service Profile UCP-Ctrl-2 (org-root/ls-UCP-Ctrl-2) [Server: sys/chassis-1/blade-1]
Service Profile UCP-Ctrl-3 (org-root/ls-UCP-Ctrl-3) [Server: sys/chassis-1/blade-2]
Are you sure you want to apply the changes?
Press Yes to disregard the warning and submit changes, No to quit the wizard or Cancel to make changes to the current configuration.
Yes
No
Cancel
Server Pool Qualification: all-chassis
OK
Cancel

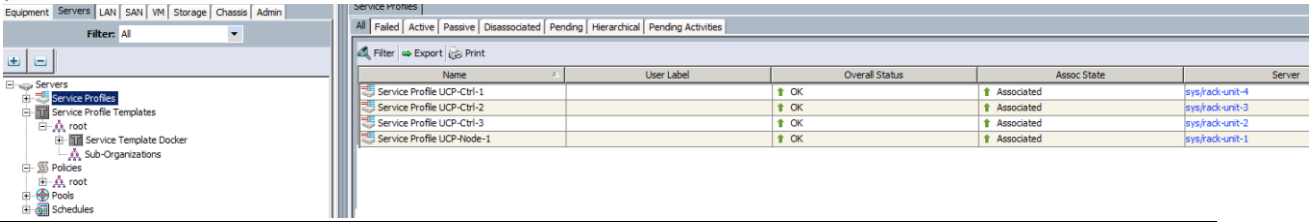




8. After the server pool association, you can view all the services profiles shown in associated state as shown in below.



For second architecture only one common template (for example, Docker) is created and service-profile are instantiated on all the four nodes:

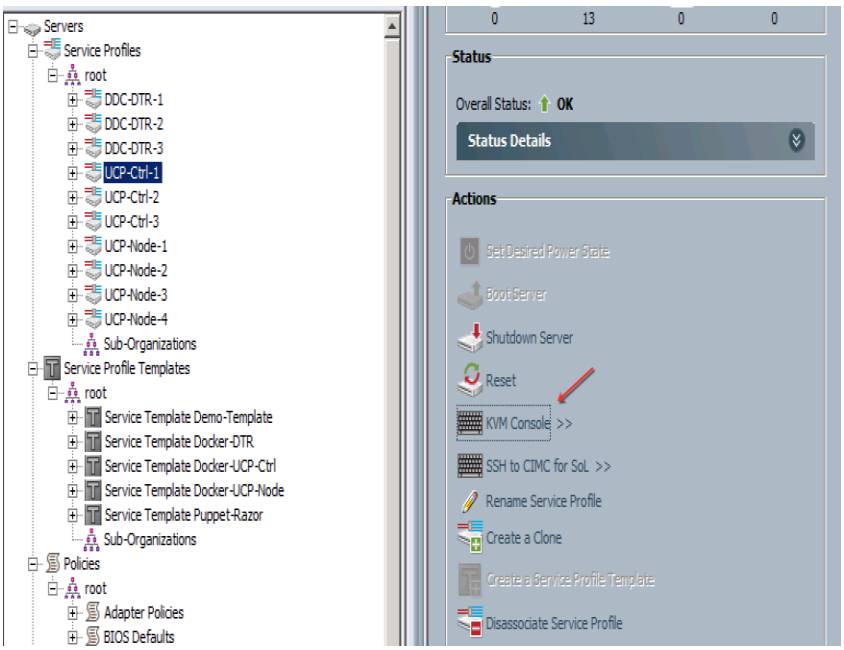


Installation of Red Hat Enterprise Linux Operating System

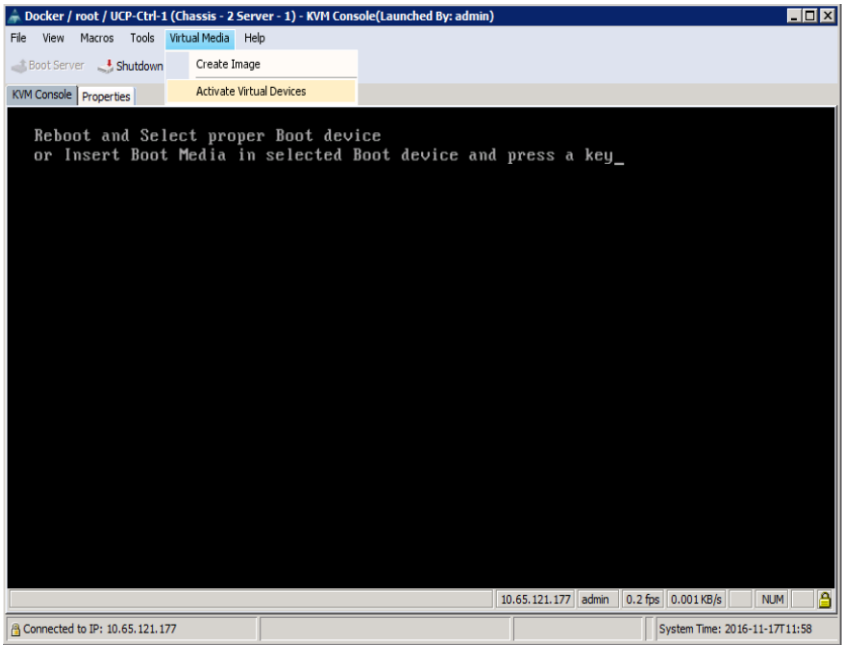
After the service profile association, Red Hat Enterprise Linux 7.2 Operating System is installed on all the blade servers. The following section provides detailed procedure for installing Red Hat Linux 7.2 OS on the Boot-Lun created using UCSM Storage Profile.

Complete the following to install Red Hat Enterprise Linux 7.2 OS.

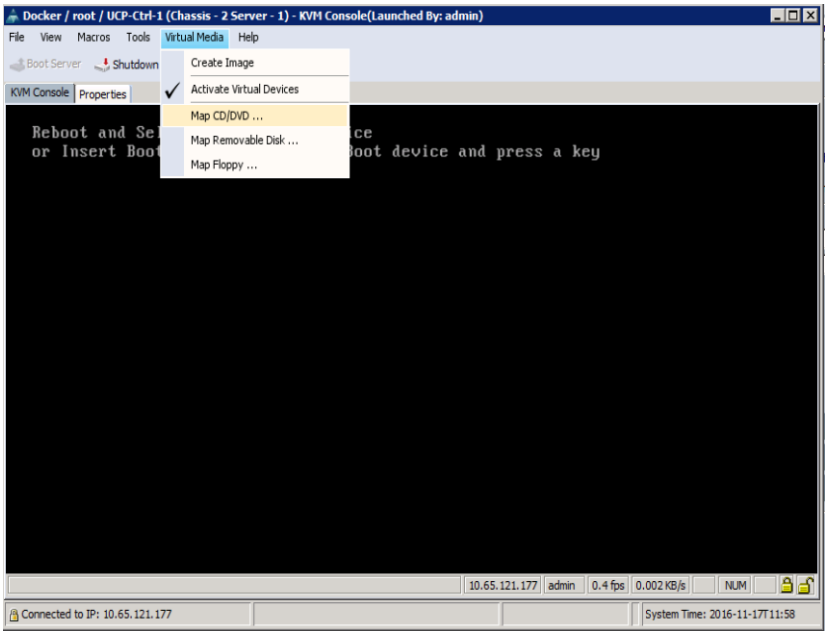
1. From Cisco UCS Manager, click Servers tab in the navigation pane.
2. Expand Servers > Service Profile Templates.
3. Select a node (for example, UCP-Ctrl-1) and click KVM Console.



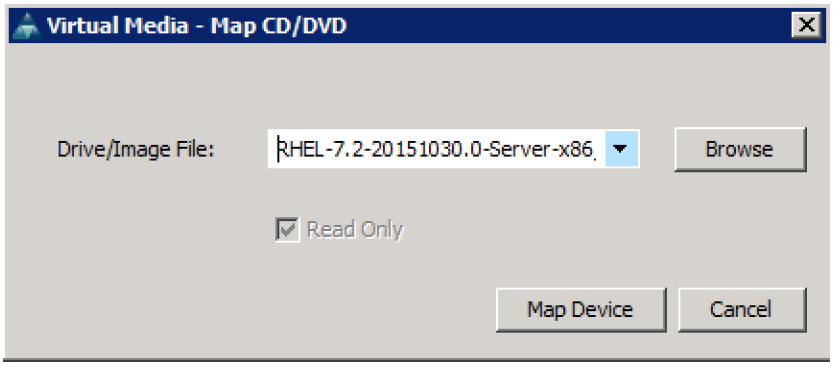
4. In the KVM Console select Virtual Media tab. From the drop-down menu select Activate virtual Devices.



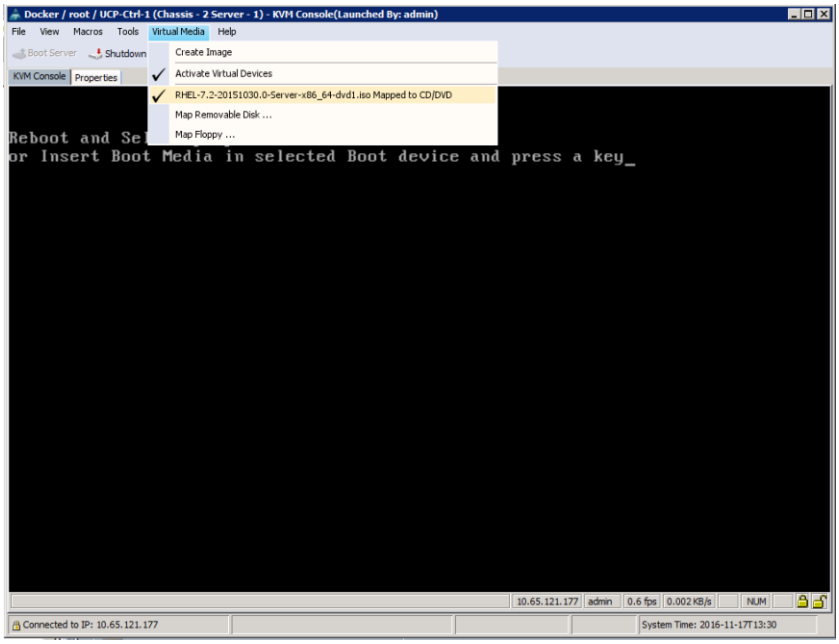
5. Once the virtual devices are activated, select Map CD/DVD option.



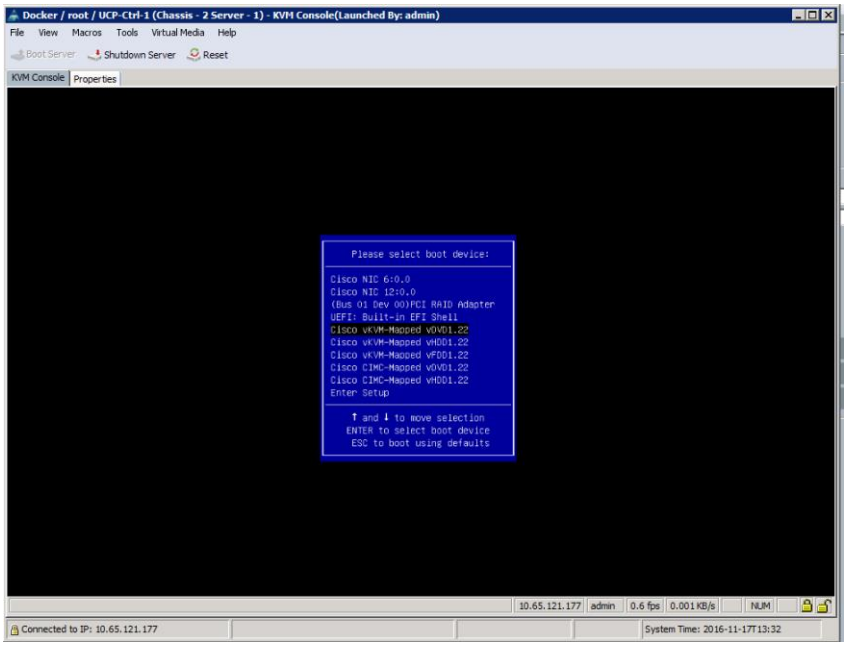
6. Click Browse to locate the Red Hat Enterprise Linux Server 7.2 installer ISO image file.



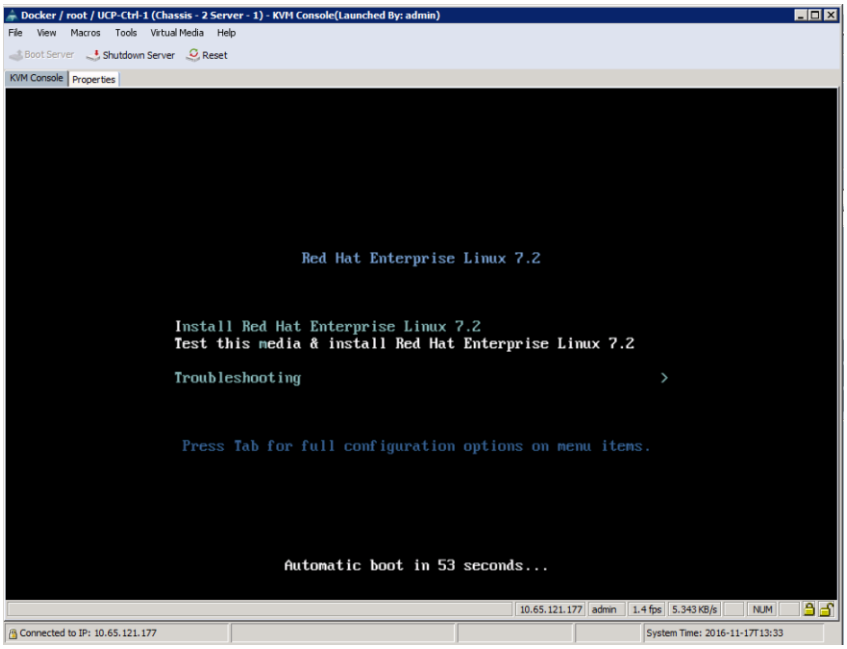
7. The image gets mapped to CD/DVD.



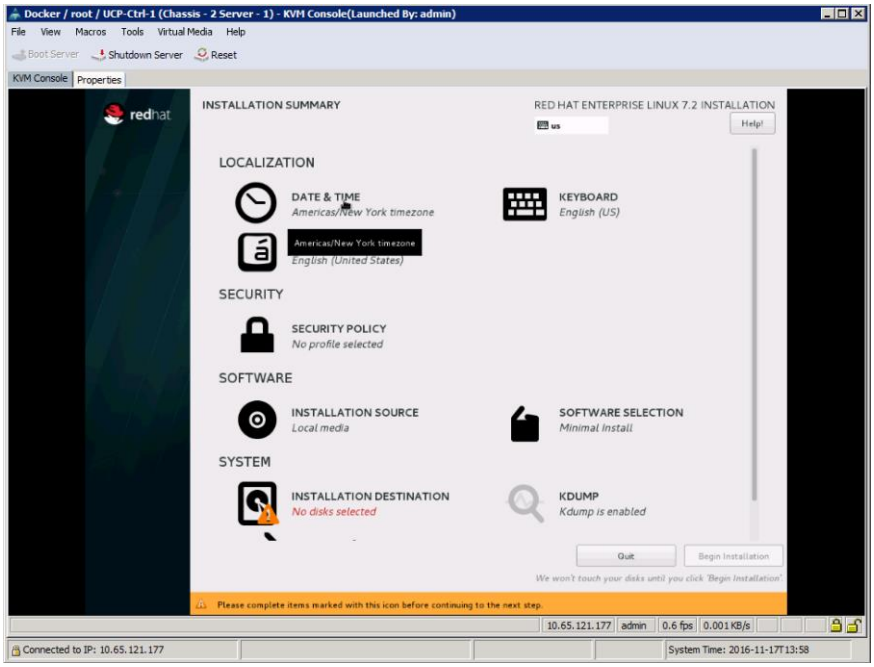
8. In the KVM window, select the KVM tab to monitor during boot.
9. In the KVM window, select the Macros > Static Macros > Ctrl-Alt-Del button in the upper left corner.
10. Click OK.
11. Click OK to reboot the system.
12. On reboot, the machine detects the presence of the Red Hat Enterprise Linux Server 7.2 install media. At the prompt press F6 to select the Boot Device. Select Cisco vKVM-Mapped vDVD.

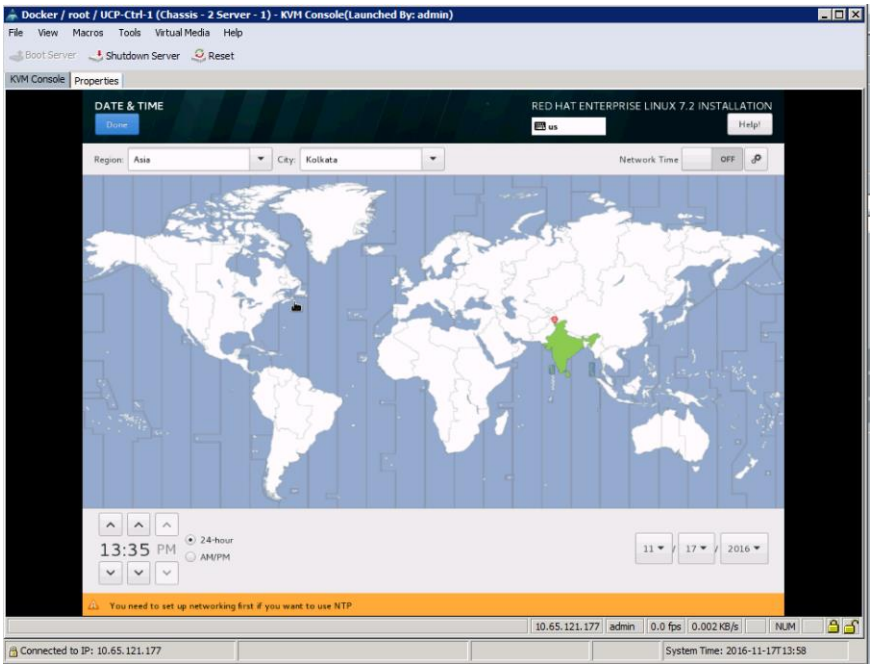


13. Select the option - Install Red Hat Enterprise Linux 7.2.

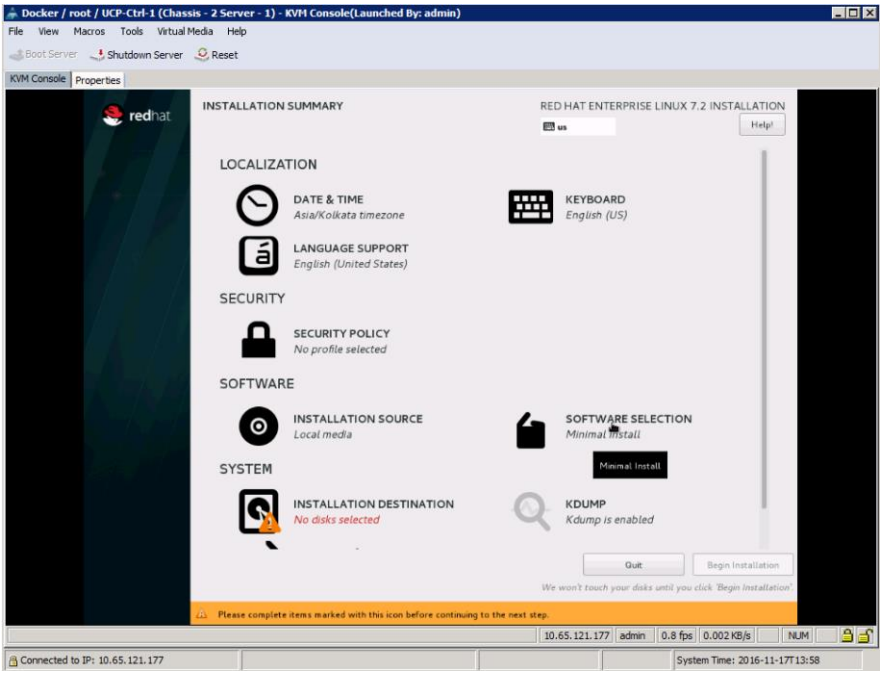


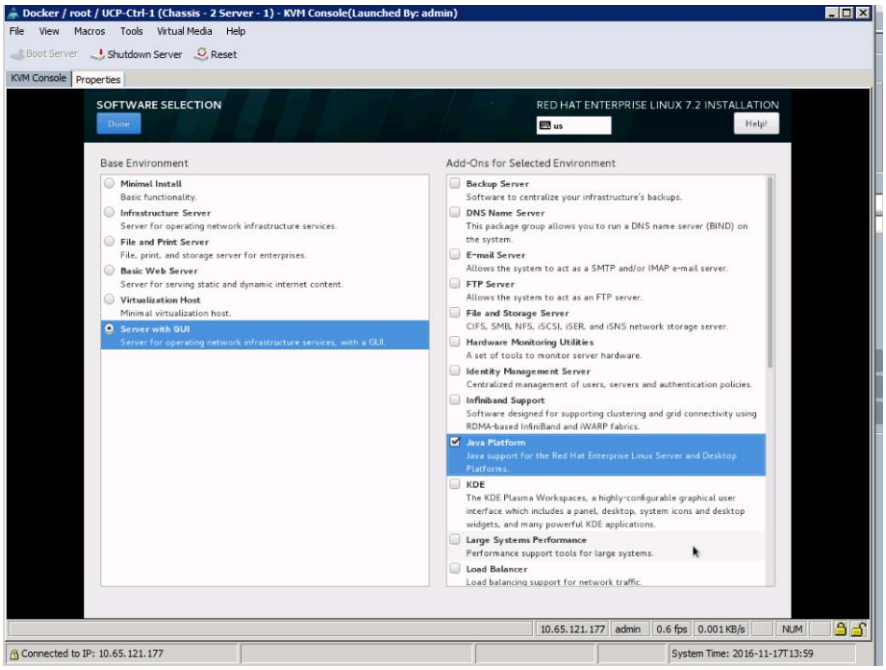
14. Click Date and Time and set the local time. And click Done.

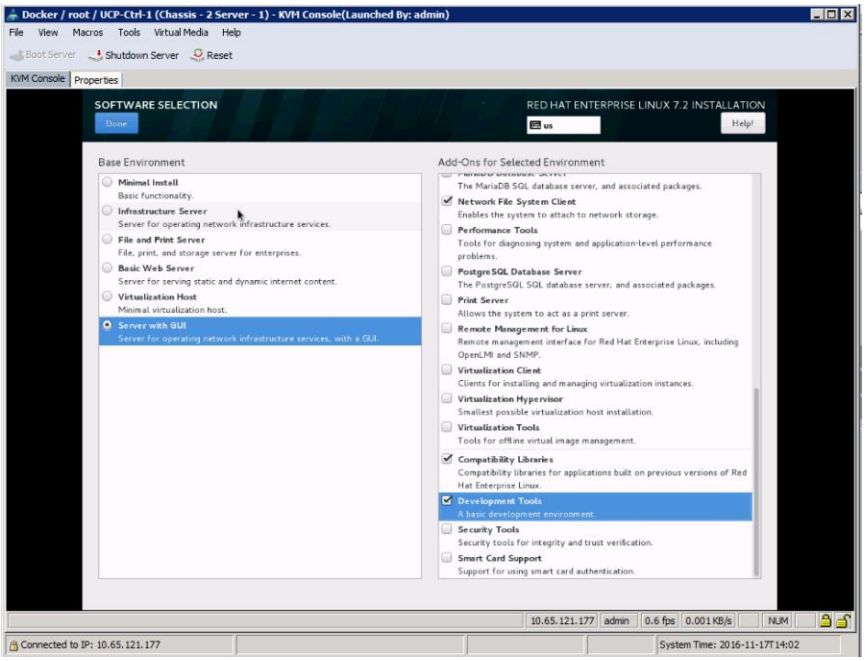




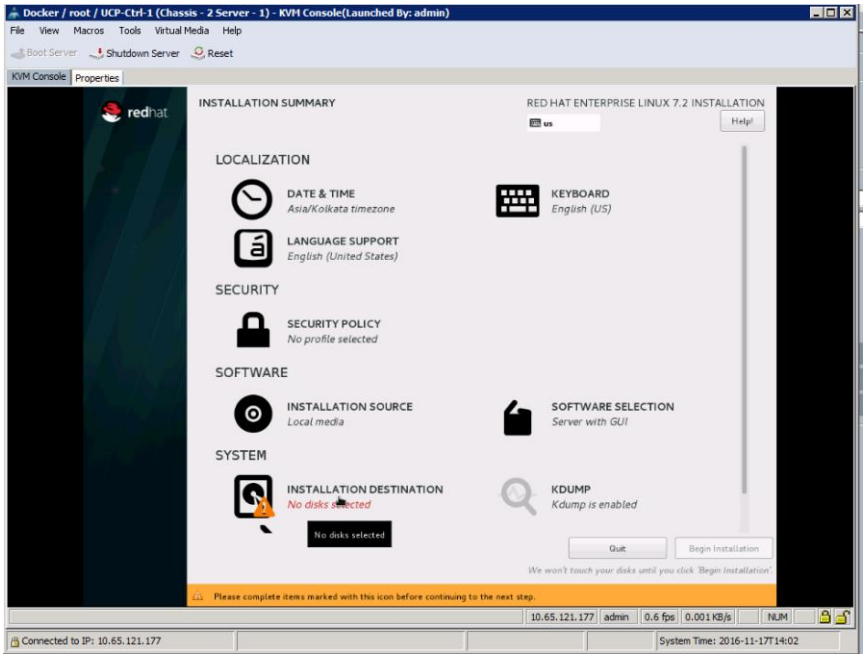
15. Click Software Selection. Under Base Environment, select the option Server with GUI. Select the necessary add-ons for the selected environment. Click Done.

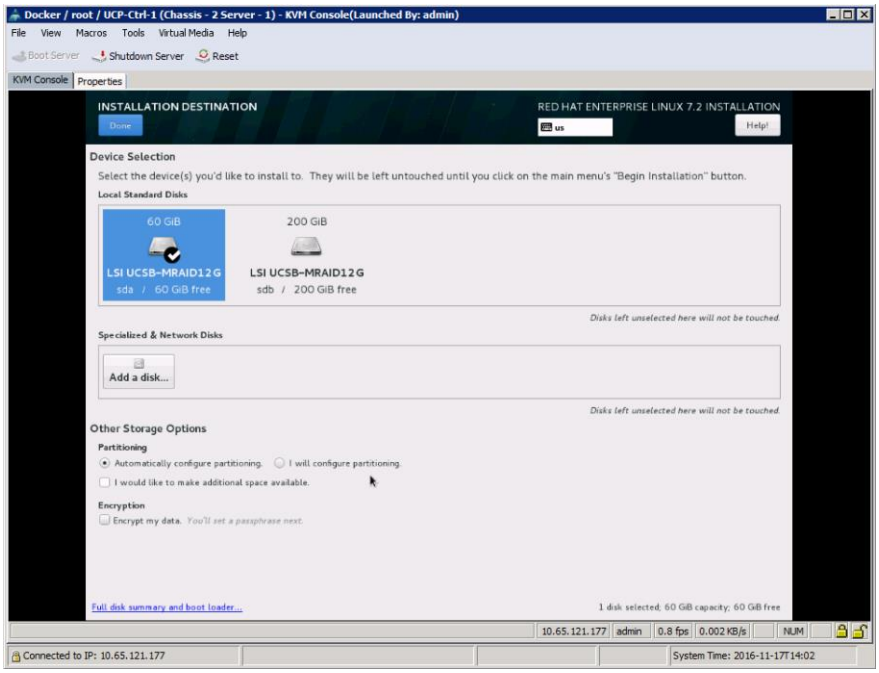

Docker / root / UCP-Ctrl-1 (Chassis - 2 Server - 1) - KVM Console(Launched By: admin)
File View Macros Tools Virtual Media Help
Boot Server Shutdown Server Reset
KVM Console Properties
redhat
INSTALLATION SUMMARY
RED HAT ENTERPRISE LINUX 7.2 INSTALLATION
us
Help!
LOCALIZATION
DATE & TIME
Asia/Kolkata timezone
KEYBOARD
English (US)
LANGUAGE SUPPORT
English (United States)
SECURITY
SECURITY POLICY
No profile selected
SOFTWARE
INSTALLATION SOURCE
Local media
SOFTWARE SELECTION
Minimal install
Minimal Install
SYSTEM
INSTALLATION DESTINATION
No disks selected
KDUMP
Kdump is enabled
Quit
Begin Installation
We won't touch your disks until you click 'Begin Installation'.
Please complete items marked with this icon before continuing to the next step.
10.65.121.177 admin 0.8 fps 0.002 KB/s NUM
Connected to IP: 10.65.121.177
System Time: 2016-11-17T13:58


Docker / root / UCP-Ctrl-1 (Chassis - 2 Server - 1) - KVM Console(Launched By: admin)
File View Macros Tools Virtual Media Help
Boot Server Shutdown Server Reset
KVM Console Properties
SOFTWARE SELECTION
Done
RED HAT ENTERPRISE LINUX 7.2 INSTALLATION
us
Help!
Base Environment
Minimal Install
Basic functionality.
Infrastructure Server
Server for operating network infrastructure services.
File and Print Server
File, print, and storage server for enterprises.
Basic Web Server
Server for serving static and dynamic internet content.
Virtualization Host
Minimal virtualization host.
Server with GUI
Server for operating network infrastructure services, with a GUI.
Add-Ons for Selected Environment
Backup Server
Software to centralize your infrastructure's backups.
DNS Name Server
This package group allows you to run a DNS name server (BIND) on the system.
E-mail Server
Allows the system to act as a SMTP and/or IMAP e-mail server.
FTP Server
Allows the system to act as an FTP server.
File and Storage Server
CIFS, SMB, NFS, iSCSI, iSER, and iSNS network storage server.
Hardware Monitoring Utilities
A set of tools to monitor server hardware.
Identity Management Server
Centralized management of users, servers and authentication policies.
Infiniband Support
Software designed for supporting clustering and grid connectivity using RDMA-based InfiniBand and iWARP fabrics.
Java Platform
Java support for the Red Hat Enterprise Linux Server and Desktop Platforms.
KDE
The KDE Plasma Workspaces, a highly-configurable graphical user interface which includes a panel, desktop, system icons and desktop widgets, and many powerful KDE applications.
Large Systems Performance
Performance support tools for large systems.
Load Balancer
Load balancing support for network traffic.
10.65.121.177 admin 0.6 fps 0.001 KB/s NUM
Connected to IP: 10.65.121.177
System Time: 2016-11-17T13:59

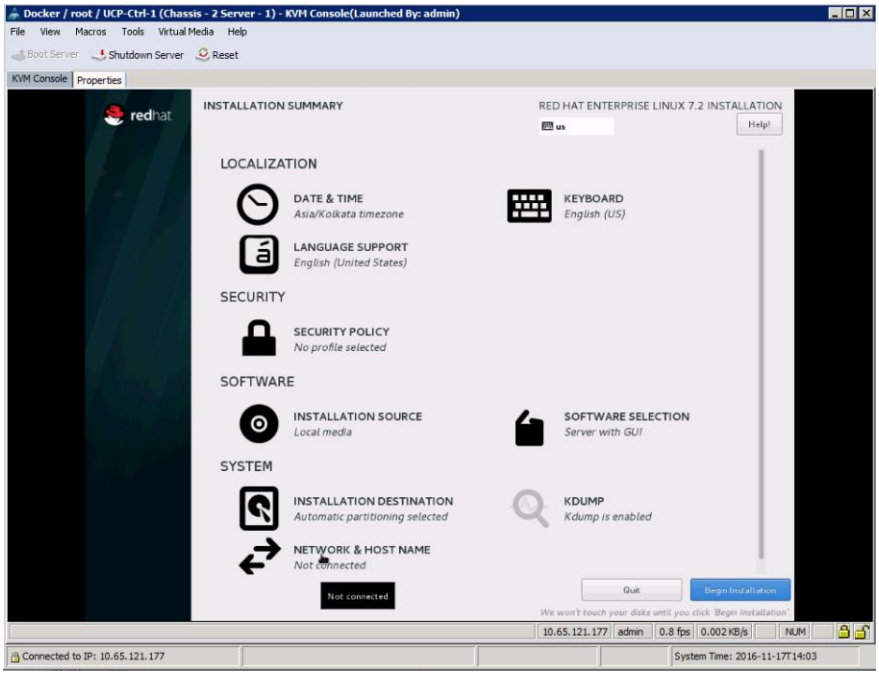


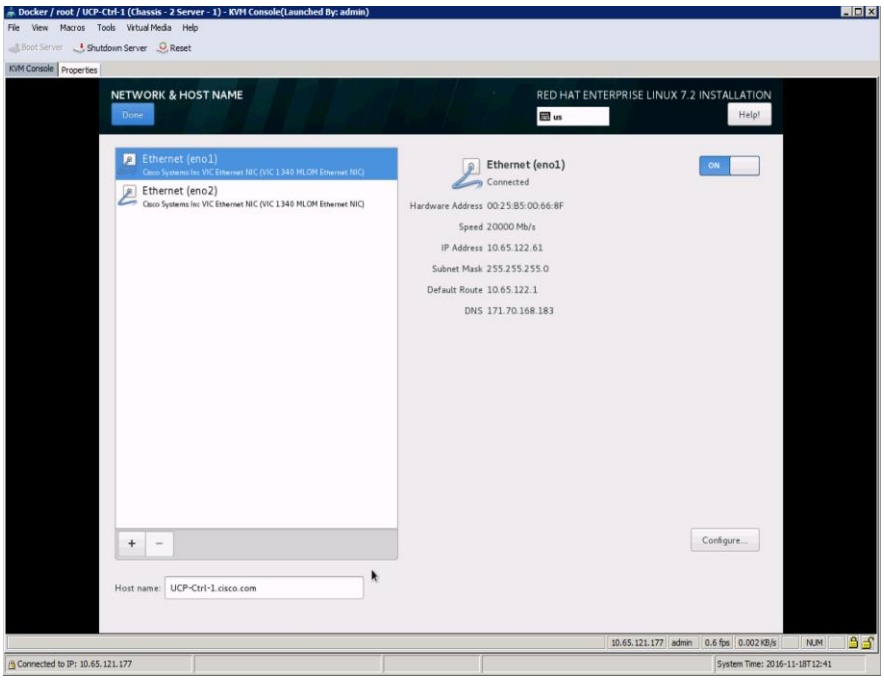
16. Click Installation Destination. In the Installation Destination window select the 60GB Lun for installing OS. Click Done.



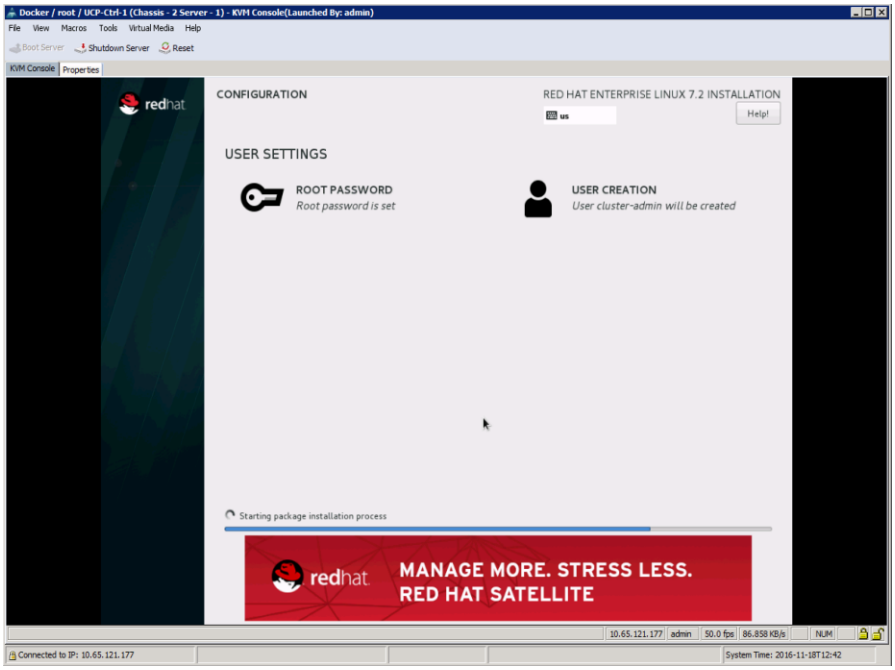


17. Click Network and Host. Select the Ethernet interface eno1 and enter the host name (for example, UCP-Ctrl-1.cisco.com). Click Configure.. and enter the IPv4 address, subnet mask and Gateway, and DNS name.

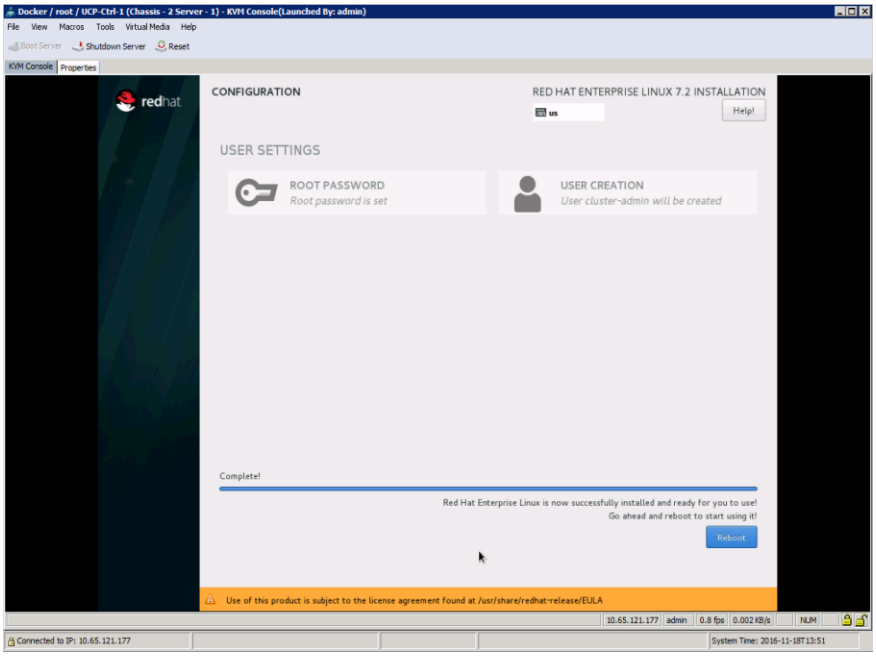




18. Click Done. And click Begin Installation.
19. While the installation in is progress, click Root Password to assert a password and then click User Creation to set user credentials.



20. Once the installation is complete, upon prompting click Reboot.



Docker Datacenter Installation

Docker Datacenter is a software subscription that includes 3 products:

- Docker CS Engine
- Docker Trusted Registry
- Docker Universal Control Plane

Both Docker UCP and DTR can be installed on-premises or on cloud as part of the installation of Docker Datacenter (DDC) on Cisco UCS blades running a baremetal operating system. Operating system does not need any additional libraries or software stack. Stock OS is sufficient to start and all the required software bits get installed as part of Docker Engine installation.

In the second architecture we have co-located the UCP Controller and DTR services together. We need to run UCP services on a TCP port other than 443. As the DTR services also use the same port for accessing its dashboard. In this alternative UCP is installed with 4443 and DTR will remain on default port 443.

There are mainly 3 steps to bring up the entire DDC on a cluster of baremetal hosts. We are required to install OS followed by Docker CS Engine and DDC. Support matrix for DDC includes RHEL 7.2 release. We have used the same in this solution.

We have used root login for the entire workflow for Docker Datacenter platform bring-up. This can also be done using ‘sudo’.

Registering Nodes and Updating Host OS

1. After the first reboot of the nodes we need to subscribe the nodes to Red Hat Subscription Manager (RHSM) for getting the latest operating system software updates. This step requires a valid subscription license with Red Hat. These steps are to be performed on all the cluster nodes.

```
[root@UCP-Ctrl-1 ~]# subscription-manager register
Registering to: subscription.rhn.redhat.com:443/subscription
Username:  < RHSM User_Name >
Password:
The system has been registered with ID: 629fadda-0b51-4a0a-9e4a-809bee25a480
```

2. Attach nodes to Red Hat Enterprise Linux Server product pool:

```
[root@UCP-Ctrl-1 ~]# subscription-manager attach --pool=  < Enter RHEL Pool ID assigned to you >
Successfully attached a subscription for: Red Hat Enterprise Linux Server, Premium (Physical or Virtual Nodes)
[root@UCP-Ctrl-1 ~]# subscription-manager release --list
+-------------------------------------------+
          Available Releases
+-------------------------------------------+
7.0
7.1
7.2
7.3
7Server
[root@UCP-Ctrl-1 ~]# subscription-manager release --set=7.2
Release set to: 7.2
```

In order to fix the updates of the operating system software to 7.2 release, we need to configure subscription-manager release set to 7.2, as noted above.

3. Verify that the release is set to a specific version of the operating system and disable all the software repos that gets attached automatically as part of system subscription. We need to enable only a specific set of repos for the stack:

```
[root@UCP-Ctrl-1 ~]# subscription-manager release --show
Release: 7.2
[root@UCP-Ctrl-1 ~]# subscription-manager repos --disable=*
Repository 'rhel-7-server-dotnet-debug-rpms' is disabled for this system.
Repository 'rhel-7-server-satellite-tools-6.2-rpms' is disabled for this system.
Repository 'rhel-7-server-v2vwin-1-debug-rpms' is disabled for this system.
Repository 'rhel-rs-for-rhel-7-server-eus-source-rpms' is disabled for this system.
Repository 'rhel-server-rhscl-7-eus-source-rpms' is disabled for this system.
Repository 'rhel-7-server-eus-rh-common-rpms' is disabled for this system.
Repository 'rhel-rs-for-rhel-7-server-eus-rpms' is disabled for this system.
Repository 'rhel-7-server-rhceph-2-tools-rpms' is disabled for this system.
Repository 'rhel-7-server-eus-debug-rpms' is disabled for this system.
Repository 'rhel-7-server-extras-rpms' is disabled for this system.
```

4. Enable specific software repos – rhel-7-server-rpms, rhel-7-server-optional-rpms & rhel-7-server-extras-rpms. This step is followed by an ‘yum update’ on all the nodes:

```
[root@UCP-Ctrl-1 ~]# subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-optional-rpms --enable=rhel-7-server-extras-rpms
Repository 'rhel-7-server-rpms' is enabled for this system.
Repository 'rhel-7-server-optional-rpms' is enabled for this system.
Repository 'rhel-7-server-extras-rpms' is enabled for this system.
[root@UCP-Ctrl-1 ~]# yum update -y
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
rhel-7-server-extras-rpms                                                | 3.4 kB  00:00:00
rhel-7-server-optional-rpms                                              | 3.3 kB  00:00:00
rhel-7-server-rpms                                                       | 3.5 kB  00:00:00
(1/9): rhel-7-server-extras-rpms/x86_64/group                            |  104 B  00:00:01
(2/9): rhel-7-server-extras-rpms/x86_64/updateinfo                       | 107 kB  00:00:02
(3/9): rhel-7-server-optional-rpms/x86_64/group                          |  25 kB  00:00:02
(4/9): rhel-7-server-extras-rpms/x86_64/primary_db                       | 146 kB  00:00:02
(5/9): rhel-7-server-optional-rpms/x86_64/updateinfo                     | 1.0 MB  00:00:09
(6/9): rhel-7-server-rpms/x86_64/group                                   | 700 kB  00:00:07
(7/9): rhel-7-server-rpms/x86_64/updateinfo                              | 1.4 MB  00:00:12
(8/9): rhel-7-server-optional-rpms/x86_64/primary_db                     | 4.2 MB  00:00:33
(9/9): rhel-7-server-rpms/x86_64/primary_db                              |  28 MB  00:03:34
Resolving Dependencies
--> Running transaction check
---> Package NetworkManager.x86_64 1:1.0.6-27.el7 will be updated
---> Package NetworkManager.x86_64 1:1.0.6-31.el7_2 will be an update
---> Package NetworkManager-adsl.x86_64 1:1.0.6-27.el7 will be updated
---> Package NetworkManager-adsl.x86_64 1:1.0.6-31.el7_2 will be an update
---> Package NetworkManager-config-server.x86_64 1:1.0.6-27.el7 will be updated
---> Package NetworkManager-config-server.x86_64 1:1.0.6-31.el7_2 will be an update
```

```
Transaction Summary
================================================================================================================================================================
Install    2 Packages
Upgrade  223 Packages

Total download size: 301 M
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
warning: /var/cache/yum/x86_64/7Server/rhel-7-server-rpms/packages/NetworkManager-adsl-1.0.6-31.el7_2.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID fd431d51: NOKEY  00:31:19 ETA
Public key for NetworkManager-adsl-1.0.6-31.el7_2.x86_64.rpm is not installed
(1/225): NetworkManager-adsl-1.0.6-31.el7_2.x86_64.rpm                                                                                  | 131 kB  00:00:02
(2/225): NetworkManager-config-server-1.0.6-31.el7_2.x86_64.rpm                                                                         | 122 kB  00:00:02
(3/225): NetworkManager-glib-1.0.6-31.el7_2.x86_64.rpm                                                                                  | 385 kB  00:00:04
(4/225): NetworkManager-libnm-1.0.6-31.el7_2.x86_64.rpm                                                                                 | 498 kB  00:00:05
(5/225): NetworkManager-1.0.6-31.el7_2.x86_64.rpm                                                                                       | 2.0 MB  00:00:17
(6/225): NetworkManager-team-1.0.6-31.el7_2.x86_64.rpm                                                                                  | 133 kB  00:00:02
(7/225): NetworkManager-tui-1.0.6-31.el7_2.x86_64.rpm                                                                                   | 212 kB  00:00:03
(8/225): NetworkManager-wifi-1.0.6-31.el7_2.x86_64.rpm                                                                                  | 160 kB  00:00:02
```

```
  gnome-terminal.x86_64 0:3.14.3-3.el7_2.1                gnutls.x86_64 0:3.3.8-14.el7_2                  gnutls-dane.x86_64 0:3.3.8-14.el7_2
  gnutls-utils.x86_64 0:3.3.8-14.el7_2                    graphite2.x86_64 0:1.3.6-1.el7_2                grub2.x86_64 1:2.02-0.34.el7_2
  grub2-tools.x86_64 1:2.02-0.34.el7_2                    gssproxy.x86_64 0:0.4.1-8.el7_2                 hpijs.x86_64 1:3.13.7-6.el7_2.1
  hplip-common.x86_64 0:3.13.7-6.el7_2.1                  hplip-libs.x86_64 0:3.13.7-6.el7_2.1            initscripts.x86_64 0:9.49.30-1.el7_2.3
  iproute.x86_64 0:3.10.0-54.el7_2.1                      ipxe-roms-qemu.noarch 0:20130517-8.gitc4bce43.el7_2.1  iscsi-initiator-utils.x86_64 0:6.2.0.873-33.el7_2.2
  iscsi-initiator-utils-iscsiuio.x86_64 0:6.2.0.873-33.el7_2.2  java-1.7.0-openjdk.x86_64 1:1.7.0.111-2.6.7.2.el7_2  java-1.7.0-openjdk-headless.x86_64 1:1.7.0.111-2.6.7.2.el7_2
  java-1.8.0-openjdk.x86_64 1:1.8.0.111-1.b15.el7_2       java-1.8.0-openjdk-headless.x86_64 1:1.8.0.111-1.b15.el7_2  kernel-headers.x86_64 0:3.10.0-327.36.3.el7
  kernel-tools.x86_64 0:3.10.0-327.36.3.el7               kernel-tools-libs.x86_64 0:3.10.0-327.36.3.el7  kexec-tools.x86_64 0:2.0.7-38.el7_2.1
  kmod.x86_64 0:20-8.el7_2                                kmod-libs.x86_64 0:20-8.el7_2                   kpartx.x86_64 0:0.4.9-85.el7_2.6
  kpatch.noarch 0:0.3.1-1.el7_2                           krb5-libs.x86_64 0:1.13.2-12.el7_2              libarchive.x86_64 0:3.1.2-10.el7_2
  libblkid.x86_64 0:2.23.2-26.el7_2.3                     libcacard.x86_64 10:1.5.3-105.el7_2.7           libgovirt.x86_64 0:0.3.3-1.el7_2.4
  libgudev1.x86_64 0:219-19.el7_2.13                      libldb.x86_64 0:1.1.25-1.el7_2                  libmount.x86_64 0:2.23.2-26.el7_2.3
  libndp.x86_64 0:1.2-6.el7_2                             libpng.x86_64 2:1.5.13-7.el7_2                  libpng12.x86_64 0:1.2.50-7.el7_2
  libreport.x86_64 0:2.1.11-31.el7                        libreport-anaconda.x86_64 0:2.1.11-31.el7       libreport-cli.x86_64 0:2.1.11-31.el7
  libreport-filesystem.x86_64 0:2.1.11-31.el7             libreport-gtk.x86_64 0:2.1.11-31.el7            libreport-plugin-bugzilla.x86_64 0:2.1.11-31.el7
  libreport-plugin-mailx.x86_64 0:2.1.11-31.el7           libreport-plugin-reportuploader.x86_64 0:2.1.11-31.el7  libreport-plugin-rhtsupport.x86_64 0:2.1.11-31.el7
  libreport-plugin-ureport.x86_64 0:2.1.11-31.el7         libreport-python.x86_64 0:2.1.11-31.el7         libreport-rhel.x86_64 0:2.1.11-31.el7
  libreport-rhel-anaconda-bugzilla.x86_64 0:2.1.11-31.el7  libreport-web.x86_64 0:2.1.11-31.el7           libreswan.x86_64 0:3.15-5.el7_1
  libsane-hpaio.x86_64 0:3.13.7-6.el7_2.1                 libsmbclient.x86_64 0:4.2.10-7.el7_2            libssh2.x86_64 0:1.4.3-10.el7_2.1
  libsss_idmap.x86_64 0:1.13.0-40.el7_2.12                libsss_nss_idmap.x86_64 0:1.13.0-40.el7_2.12    libtalloc.x86_64 0:2.1.5-1.el7_2
  libtdb.x86_64 0:1.3.8-1.el7_2                           libteam.x86_64 0:1.17-7.el7_2                   libtevent.x86_64 0:0.9.26-1.el7_2.1
  libtiff.x86_64 0:4.0.3-25.el7_2                         libtool.x86_64 0:2.4.2-21.el7_2                 libtool-ltdl.x86_64 0:2.4.2-21.el7_2
  libunwind.x86_64 2:1.1-5.el7_2.2                        libuuid.x86_64 0:2.23.2-26.el7_2.3              libvirt-client.x86_64 0:1.2.17-13.el7_2.5
  libvirt-daemon.x86_64 0:1.2.17-13.el7_2.5               libvirt-daemon-config-network.x86_64 0:1.2.17-13.el7_2.5  libvirt-daemon-driver-interface.x86_64 0:1.2.17-13.el7_2.5
  libvirt-daemon-driver-network.x86_64 0:1.2.17-13.el7_2.5  libvirt-daemon-driver-nodedev.x86_64 0:1.2.17-13.el7_2.5  libvirt-daemon-driver-nwfilter.x86_64 0:1.2.17-13.el7_2.5
  libvirt-daemon-driver-qemu.x86_64 0:1.2.17-13.el7_2.5   libvirt-daemon-driver-secret.x86_64 0:1.2.17-13.el7_2.5  libvirt-daemon-driver-storage.x86_64 0:1.2.17-13.el7_2.5
  libvirt-daemon-kvm.x86_64 0:1.2.17-13.el7_2.5           libwbclient.x86_64 0:4.2.10-7.el7_2             libxml2.x86_64 0:2.9.1-6.el7_2.3
  libxml2-python.x86_64 0:2.9.1-6.el7_2.3                 logrotate.x86_64 0:3.8.6-7.el7_2                lvm2.x86_64 7:2.02.130-5.el7_2.5
  lvm2-libs.x86_64 7:2.02.130-5.el7_2.5                   mariadb-libs.x86_64 1:5.5.50-1.el7_2            mdadm.x86_64 0:3.3.2-7.el7_2.1
  microcode_ctl.x86_64 2:2.1-12.el7_2.1                   net-snmp-libs.x86_64 1:5.7.2-24.el7_2.1         nfs-utils.x86_64 1:1.3.0-0.21.el7_2.1
  nspr.x86_64 0:4.11.0-1.el7_2                            nss.x86_64 0:3.21.0-9.el7_2                     nss-softokn.x86_64 0:3.16.2.3-14.2.el7_2
  nss-softokn-freebl.x86_64 0:3.16.2.3-14.2.el7_2         nss-sysinit.x86_64 0:3.21.0-9.el7_2             nss-tools.x86_64 0:3.21.0-9.el7_2
  nss-util.x86_64 0:3.21.0-2.2.el7_2                      ntpdate.x86_64 0:4.2.6p5-22.el7_2.2             ntsysv.x86_64 0:1.3.61-5.el7_2.1
  numactl-libs.x86_64 0:2.0.9-6.el7_2                     open-vm-tools.x86_64 0:9.10.2-5.el7_2           open-vm-tools-desktop.x86_64 0:9.10.2-5.el7_2
  openldap.x86_64 0:2.4.40-9.el7_2                        openssh.x86_64 0:6.6.1p1-25.el7_2               openssh-clients.x86_64 0:6.6.1p1-25.el7_2
  openssh-server.x86_64 0:6.6.1p1-25.el7_2                openssl.x86_64 1:1.0.1e-51.el7_2.7              openssl-libs.x86_64 1:1.0.1e-51.el7_2.7
  openssl098e.x86_64 0:0.9.8e-29.el7_2.3                  pcre.x86_64 0:8.32-15.el7_2.1                   perl-Git.noarch 0:1.8.3.1-6.el7_2.1
  polkit.x86_64 0:0.112-7.el7_2                           procps-ng.x86_64 0:3.3.10-5.el7_2               pytalloc.x86_64 0:2.1.5-1.el7_2
  python.x86_64 0:2.7.5-39.el7_2                          python-libs.x86_64 0:2.7.5-39.el7_2             python-perf.x86_64 0:3.10.0-327.36.3.el7
  python-pyudev.noarch 0:0.15-7.el7_2.1                   qemu-img.x86_64 10:1.5.3-105.el7_2.7            qemu-kvm.x86_64 10:1.5.3-105.el7_2.7
  qemu-kvm-common.x86_64 10:1.5.3-105.el7_2.7             quota.x86_64 1:4.01-11.el7_2.1                  quota-nls.noarch 1:4.01-11.el7_2.1
  rdma.noarch 0:7.2_4.1_rc6-2.el7                         rpcbind.x86_64 0:0.2.0-33.el7_2.1               samba-client.x86_64 0:4.2.10-7.el7_2
  samba-client-libs.x86_64 0:4.2.10-7.el7_2               samba-common.noarch 0:4.2.10-7.el7_2            samba-common-libs.x86_64 0:4.2.10-7.el7_2
  samba-common-tools.x86_64 0:4.2.10-7.el7_2              samba-libs.x86_64 0:4.2.10-7.el7_2              selinux-policy.noarch 0:3.13.1-60.el7_2.9
  selinux-policy-targeted.noarch 0:3.13.1-60.el7_2.9      setroubleshoot.x86_64 0:3.2.24-4.el7_2          setroubleshoot-plugins.noarch 0:3.0.59-2.el7_2
  setroubleshoot-server.x86_64 0:3.2.24-4.el7_2           sos.noarch 0:3.2-35.el7_2.3                     spice-server.x86_64 0:0.12.4-15.el7_2.2
  sssd-client.x86_64 0:1.13.0-40.el7_2.12                 sudo.x86_64 0:1.8.6p7-17.el7_2                  systemd.x86_64 0:219-19.el7_2.13
  systemd-libs.x86_64 0:219-19.el7_2.13                   systemd-python.x86_64 0:219-19.el7_2.13         systemd-sysv.x86_64 0:219-19.el7_2.13
  teamd.x86_64 0:1.17-7.el7_2                             tigervnc-license.noarch 0:1.3.1-4.el7_2         tigervnc-server-minimal.x86_64 0:1.3.1-4.el7_2
  tuned.noarch 0:2.5.1-4.el7_2.6                          tzdata.noarch 0:2016h-1.el7                     tzdata-java.noarch 0:2016h-1.el7
  util-linux.x86_64 0:2.23.2-26.el7_2.3

Complete!
```

5. Once yum update is complete, reboot all the nodes.

Installing and Configuring Ansible

In order to speed up post installation tasks on a cluster consisting of 10 nodes, we have used an open source automation tool called Ansible. This powerful tool needs very few steps to configure nodes and get up and running. This tool is not based on a 'server-client' model for executing automated tasks. Ansible tool can be installed through EPEL (Extra Packages for Enterprise Linux) repos.

Ansible does not require a build host, any host with in a group of hosts subjected for post install configuration can take a role of Ansible controller node. Ansible controller node does not require any additional software or packages. It's the node from where we run Ansible commands/playbook for automated configuration of all the nodes including the controller nodes itself. Note that the controller node is also part of Docker Datacenter. Steps to install and configure Ansible are listed below:

1. Generate and populate ssh key of the Ansible controller node to itself and rest of the nodes in the cluster. This is required for password-less ssh login to all the nodes in order to execute configuration tasks.

```
[root@UCP-Ctrl-1 ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
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:
eb:93:22:2c:11:41:12:d8:f7:60:35:96:d9:87:1a:77 root@UCP-Ctrl-1.cisco.com
The key's randomart image is:
+--[ RSA 2048]----+
|++.  .++ .       |
|..o +.+.+ E      |
|   + o + o       |
|  .   o          |
|   .    S        |
|  .      .       |
|   o    ..       |
|  . o ..o        |
|   . . ...       |
+-----------------+
```

2. Using ‘ssh-copy-id’ copy key is generated to all the nodes including control node:

```
[root@UCP-Ctrl-1 ~]# ssh-copy-id root@UCP-Ctrl-1
The authenticity of host 'ucp-ctrl-1 (10.65.122.61)' can't be established.
ECDSA key fingerprint is 98:11:14:1c:5c:e2:93:fb:23:e9:d5:6f:5f:0b:2a:a1.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@ucp-ctrl-1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@UCP-Ctrl-1'"
and check to make sure that only the key(s) you wanted were added.
```

```
[root@UCP-Ctrl-1 ~]# ssh-copy-id root@UCP-Ctrl-2
The authenticity of host 'ucp-ctrl-2 (10.65.122.62)' can't be established.
ECDSA key fingerprint is d5:5b:30:59:84:37:06:56:ab:1a:b3:9a:d7:74:85:62.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@ucp-ctrl-2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@UCP-Ctrl-2'"
and check to make sure that only the key(s) you wanted were added.

[root@UCP-Ctrl-1 ~]# ssh-copy-id root@UCP-Ctrl-3
The authenticity of host 'ucp-ctrl-3 (10.65.122.63)' can't be established.
ECDSA key fingerprint is bc:00:dd:ac:72:22:30:fd:9b:46:12:9d:02:f5:14:44.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@ucp-ctrl-3's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@UCP-Ctrl-3'"
and check to make sure that only the key(s) you wanted were added.
```

3. Install epel rpm, for which we have to download latest the ‘epel’ rpm for attaching epel repos to all the cluster nodes:

```
[root@UCP-Ctrl-1 ~]# ls -ltr
total 24
-rw-------. 1 root root  1852 Nov 18 12:46 anaconda-ks.cfg
-rw-------. 1 root root  1945 Nov 18 13:34 initial-setup-ks.cfg
-rw-r--r--. 1 root root 14612 Nov 19 22:51 epel-release-latest-7.noarch.rpm
[root@UCP-Ctrl-1 ~]# rpm -ivh epel-release-latest-7.noarch.rpm
warning: epel-release-latest-7.noarch.rpm: Header V3 RSA/SHA256 Signature, key ID 352c64e5: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:epel-release-7-8                 ################################# [100%]
[root@UCP-Ctrl-1 ~]#
```

epel rpm can be downloaded from - https://fedoraproject.org/wiki/EPEL

4. Copy the rpms on the remaining nodes and install:

```
[root@UCP-Ctrl-1 ~]# for i in 62 63 64 65 67 68 69 70;do echo 10.65.122.$i;scp epel-release-latest-7.noarch.rpm 10.65.122.$i:/root/;done
10.65.122.62
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.63
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.64
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.65
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.67
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.68
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.69
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
10.65.122.70
epel-release-latest-7.noarch.rpm                                                                    100%   14KB  14.3KB/s   00:00
```

5. Check for the available Ansible version, we have used version 2.2:

```
[root@UCP-Ctrl-1 ~]# yum info ansible
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
Available Packages
Name        : ansible
Arch        : noarch
Version     : 2.2.0.0
Release     : 3.el7
Size        : 4.6 M
Repo        : epel/x86_64
Summary     : SSH-based configuration management, deployment, and task execution system
URL         : http://ansible.com
License     : GPLv3+
Description :
            : Ansible is a radically simple model-driven configuration management,
            : multi-node deployment, and remote task execution system. Ansible works
            : over SSH and does not require any software or daemons to be installed
            : on remote nodes. Extension modules can be written in any language and
            : are transferred to managed machines automatically.
```

6. Install Ansible using yum on all the nodes:

```
[root@UCP-Ctrl-1 ~]# yum install ansible
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package ansible.noarch 0:2.2.0.0-3.el7 will be installed
--> Processing Dependency: sshpass for package: ansible-2.2.0.0-3.el7.noarch
--> Processing Dependency: python-paramiko for package: ansible-2.2.0.0-3.el7.noarch
--> Processing Dependency: python-keyczar for package: ansible-2.2.0.0-3.el7.noarch
--> Processing Dependency: python-jinja2 for package: ansible-2.2.0.0-3.el7.noarch
--> Processing Dependency: python-httplib2 for package: ansible-2.2.0.0-3.el7.noarch
--> Processing Dependency: PyYAML for package: ansible-2.2.0.0-3.el7.noarch
--> Running transaction check
---> Package PyYAML.x86_64 0:3.10-11.el7 will be installed
--> Processing Dependency: libyaml-0.so.2()(64bit) for package: PyYAML-3.10-11.el7.x86_64
---> Package python-httplib2.noarch 0:0.7.7-3.el7 will be installed
---> Package python-jinja2.noarch 0:2.7.2-2.el7 will be installed
--> Processing Dependency: python-babel >= 0.8 for package: python-jinja2-2.7.2-2.el7.noarch
--> Processing Dependency: python-markupsafe for package: python-jinja2-2.7.2-2.el7.noarch
---> Package python-keyczar.noarch 0:0.71c-2.el7 will be installed
--> Processing Dependency: python-pyasn1 for package: python-keyczar-0.71c-2.el7.noarch
--> Processing Dependency: python-crypto for package: python-keyczar-0.71c-2.el7.noarch
---> Package python2-paramiko.noarch 0:1.16.1-1.el7 will be installed
--> Processing Dependency: python2-ecdsa for package: python2-paramiko-1.16.1-1.el7.noarch
---> Package sshpass.x86_64 0:1.05-5.el7 will be installed
--> Running transaction check
---> Package libyaml.x86_64 0:0.1.4-11.el7_0 will be installed
---> Package python-babel.noarch 0:0.9.6-8.el7 will be installed
---> Package python-markupsafe.x86_64 0:0.11-10.el7 will be installed
---> Package python-pyasn1.noarch 0:0.1.6-2.el7 will be installed
---> Package python2-crypto.x86_64 0:2.6.1-9.el7 will be installed
--> Processing Dependency: libtomcrypt.so.0()(64bit) for package: python2-crypto-2.6.1-9.el7.x86_64
---> Package python2-ecdsa.noarch 0:0.13-4.el7 will be installed
--> Running transaction check
---> Package libtomcrypt.x86_64 0:1.17-23.el7 will be installed
--> Processing Dependency: libtommath >= 0.42.0 for package: libtomcrypt-1.17-23.el7.x86_64
--> Processing Dependency: libtommath.so.0()(64bit) for package: libtomcrypt-1.17-23.el7.x86_64
--> Running transaction check
---> Package libtommath.x86_64 0:0.42.0-4.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                Arch          Version                Repository      Size
================================================================================
Installing:
 ansible                noarch        2.2.0.0-3.el7          epel           4.6 M
Installing for dependencies:
 PyYAML                 x86_64        3.10-11.el7            rhel-7-server-rpms
                                                                              153 k
 libtomcrypt            x86_64        1.17-23.el7            epel           224 k
 libtommath             x86_64        0.42.0-4.el7           epel            35 k
 libyaml                x86_64        0.1.4-11.el7_0         rhel-7-server-rpms
                                                                               55 k
 python-babel           noarch        0.9.6-8.el7            rhel-7-server-rpms
                                                                              1.4 M
 python-httplib2        noarch        0.7.7-3.el7            epel            70 k
```

```
Installed:
  ansible.noarch 0:2.2.0.0-3.el7

Dependency Installed:
  PyYAML.x86_64 0:3.10-11.el7                libtomcrypt.x86_64 0:1.17-23.el7
  libtommath.x86_64 0:0.42.0-4.el7           libyaml.x86_64 0:0.1.4-11.el7_0
  python-babel.noarch 0:0.9.6-8.el7          python-httplib2.noarch 0:0.7.7-3.el7
  python-jinja2.noarch 0:2.7.2-2.el7         python-markupsafe.x86_64 0:0.11-10.el7
  python-pyasn1.noarch 0:0.1.6-2.el7         python2-crypto.x86_64 0:2.6.1-9.el7
  python2-ecdsa.noarch 0:0.13-4.el7          python2-paramiko.noarch 0:1.16.1-1.el7
  sshpass.x86_64 0:1.05-5.el7

Complete!
```

7. If the nodes does not have FQDN as hostname, we will have to update the '/etc/host' file with all cluster node entries:

```
[root@UCP-Ctrl-1 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.65.122.61    UCP-Ctrl-1.cisco.com    UCP-Ctrl-1
10.65.122.62    UCP-Ctrl-2.cisco.com    UCP-Ctrl-2
10.65.122.63    UCP-Ctrl-3.cisco.com    UCP-Ctrl-3
10.65.122.64    DDC-DTR-1.cisco.com     DDC-DTR-1
10.65.122.65    DDC-DTR-2.cisco.com     DDC-DTR-2
10.65.122.66    DDC-DTR-3.cisco.com     DDC-DTR-3
10.65.122.67    UCP-Node-1.cisco.com    UCP-Node-1
10.65.122.68    UCP-Node-2.cisco.com    UCP-Node-2
10.65.122.69    UCP-Node-3.cisco.com    UCP-Node-3
10.65.122.70    UCP-Node-4.cisco.com    UCP-Node-4
```

8. Ansible relies on the concept of host groups, all the configuration tasks are targeted to specific group of hosts. Since all our nodes are similar in nature for software stack deployment, we created a single host group 'Docker' as shown below:

```
[root@UCP-Ctrl-1 ~]# cat /etc/ansible/hosts |grep -A 14 Docker
[Docker]
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
[DTR]
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
```

To run the other tasks specific to DTR nodes we created a host group for DTR nodes, as shown above.

9. Verify Ansible is configured correctly on all the nodes by using its ICMP module, you can check if the nodes are reachable:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m ping
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
DDC-DTR-2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
DDC-DTR-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
DDC-DTR-3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Node-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Node-2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Node-3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
UCP-Node-4 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

10. Populate the /etc/hosts file on the remaining nodes from controller node:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/etc/hosts dest=/etc/hosts"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "6e8aebbfb1e06d09dfd48fe448ee89a994f530cc",
    "dest": "/etc/hosts",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/hosts",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 602,
    "state": "file",
    "uid": 0
}
DDC-DTR-2 | SUCCESS => {
    "changed": true,
    "checksum": "6e8aebbfb1e06d09dfd48fe448ee89a994f530cc",
    "dest": "/etc/hosts",
    "gid": 0,
    "group": "root",
    "md5sum": "a68470d659f0952a7fe599b2990b2e23",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 602,
    "src": "/root/.ansible/tmp/ansible-tmp-1479577631.2-119323327479829/source",
    "state": "file",
    "uid": 0
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": true,
    "checksum": "6e8aebbfb1e06d09dfd48fe448ee89a994f530cc",
    "dest": "/etc/hosts",
    "gid": 0,
    "group": "root",
    "md5sum": "a68470d659f0952a7fe599b2990b2e23",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 602,
    "src": "/root/.ansible/tmp/ansible-tmp-1479577631.2-253037475841184/source",
    "state": "file",
    "uid": 0
}
```

Installing NTP and Configuring Host OS System Clocks

1. Docker Datacenter requires nodes participating in the cluster to have their system time be in sync with the external source. We do this by installing NTP services and configuring it to remain in sync with the system time:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m yum -a "name=ntp state=present"
UCP-Ctrl-2 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nResolving Dependencies\n--> Running transaction check\n---> Package ntp.
x86_64 0:4.2.6p5-22.el7_2.2 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
=======\n Package   Arch        Version                Repository            Size\n================================================================================\nInstalling:
\n ntp       x86_64       4.2.6p5-22.el7_2.2       rhel-7-server-rpms       544 k\n\nTransaction Summary\n================================================================================
===\nInstall  1 Package\n\nTotal download size: 544 k\nInstalled size: 1.4 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nR
unning transaction\n  Installing : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \n  Verifying  : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \
n\nInstalled:\n  ntp.x86_64 0:4.2.6p5-22.el7_2.2                                              \n\nComplete!\n"
    ]
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nResolving Dependencies\n--> Running transaction check\n---> Package ntp.
x86_64 0:4.2.6p5-22.el7_2.2 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
=======\n Package   Arch        Version                Repository            Size\n================================================================================\nInstalling:
\n ntp       x86_64       4.2.6p5-22.el7_2.2       rhel-7-server-rpms       544 k\n\nTransaction Summary\n================================================================================
===\nInstall  1 Package\n\nTotal download size: 544 k\nInstalled size: 1.4 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nR
unning transaction\n  Installing : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \n  Verifying  : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \
n\nInstalled:\n  ntp.x86_64 0:4.2.6p5-22.el7_2.2                                              \n\nComplete!\n"
    ]
}
UCP-Ctrl-1 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nResolving Dependencies\n--> Running transaction check\n---> Package ntp.
x86_64 0:4.2.6p5-22.el7_2.2 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
=======\n Package   Arch        Version                Repository            Size\n================================================================================\nInstalling:
\n ntp       x86_64       4.2.6p5-22.el7_2.2       rhel-7-server-rpms       544 k\n\nTransaction Summary\n================================================================================
===\nInstall  1 Package\n\nTotal download size: 544 k\nInstalled size: 1.4 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nR
unning transaction\n  Installing : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \n  Verifying  : ntp-4.2.6p5-22.el7_2.2.x86_64                                 1/1 \
n\nInstalled:\n  ntp.x86_64 0:4.2.6p5-22.el7_2.2                                              \n\nComplete!\n"
    ]
}
```

2. Add your NTP source in /etc/ntp.conf file and copy the file to all other nodes:

```
[root@UCP-Ctrl-1 ~]# cat /etc/ntp.conf | grep server
# Use public servers from the pool.ntp.org project.
#server 0.rhel.pool.ntp.org iburst
#server 1.rhel.pool.ntp.org iburst
#server 2.rhel.pool.ntp.org iburst
#server 3.rhel.pool.ntp.org iburst
server 171.68.38.66
#broadcast 192.168.1.255 autokey        # broadcast server
#broadcast 224.0.1.1 autokey            # multicast server
#manycastserver 239.255.254.254         # manycast server
```

3. Copy the conf file to all the other nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/etc/ntp.conf dest=/etc/ntp.conf"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "de3966eeaa8a1ff585c43f9723962643719ccb85",
    "dest": "/etc/ntp.conf",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/ntp.conf",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 2016,
    "state": "file",
    "uid": 0
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": true,
    "checksum": "de3966eeaa8a1ff585c43f9723962643719ccb85",
    "dest": "/etc/ntp.conf",
    "gid": 0,
    "group": "root",
    "md5sum": "55f130d7f7361d118aaf3a1b0d8a9e3c",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 2016,
    "src": "/root/.ansible/tmp/ansible-tmp-1479578047.86-247148664869733/source",
    "state": "file",
    "uid": 0
}
DDC-DTR-1 | SUCCESS => {
    "changed": true,
    "checksum": "de3966eeaa8a1ff585c43f9723962643719ccb85",
    "dest": "/etc/ntp.conf",
    "gid": 0,
    "group": "root",
    "md5sum": "55f130d7f7361d118aaf3a1b0d8a9e3c",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:net_conf_t:s0",
    "size": 2016,
    "src": "/root/.ansible/tmp/ansible-tmp-1479578047.86-229495260626014/source",
    "state": "file",
    "uid": 0
}
```

4. Enable NTPD and start the service, followed by status check to make sure system clocks are in sync on all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl enable ntpd.service"
DDC-DTR-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

DDC-DTR-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

DDC-DTR-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Node-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Node-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Node-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.

UCP-Node-4 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/ntpd.service to /usr/lib/systemd/system/ntpd.service.
```

5. Start the NTPD service on all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl start ntpd.service"
UCP-Ctrl-1 | SUCCESS | rc=0 >>


UCP-Ctrl-3 | SUCCESS | rc=0 >>


DDC-DTR-2 | SUCCESS | rc=0 >>


DDC-DTR-1 | SUCCESS | rc=0 >>


UCP-Ctrl-2 | SUCCESS | rc=0 >>


DDC-DTR-3 | SUCCESS | rc=0 >>


UCP-Node-3 | SUCCESS | rc=0 >>


UCP-Node-1 | SUCCESS | rc=0 >>


UCP-Node-2 | SUCCESS | rc=0 >>


UCP-Node-4 | SUCCESS | rc=0 >>
```

6. Check the status of the NTPD service on all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl status ntpd.service"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Sat 2016-11-19 23:26:42 IST; 23min ago
  Process: 29098 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 29099 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─29099 /usr/sbin/ntpd -u ntp:ntp -g

Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: Listen normally on 5 lo ::1 UDP 123
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: Listen normally on 6 eno1 fe80::225:b5ff:fe00:668f UDP 123
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: Listening on routing socket on fd #23 for interface updates
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c016 06 restart
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c012 02 freq_set kernel 0.000 PPM
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c011 01 freq_not_set
Nov 19 23:26:42 UCP-Ctrl-1.cisco.com systemd[1]: Started Network Time Service.
Nov 19 23:26:44 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c61c 0c clock_step +1398.707053 s
Nov 19 23:50:02 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c614 04 freq_mode
Nov 19 23:50:03 UCP-Ctrl-1.cisco.com ntpd[29099]: 0.0.0.0 c618 08 no_sys_peer

DDC-DTR-2 | SUCCESS | rc=0 >>
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2016-11-20 23:27:25 IST; 23h left
  Process: 17376 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 17377 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─17377 /usr/sbin/ntpd -u ntp:ntp -g

Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: Listen normally on 4 virbr0 192.168.122.1 UDP 123
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: Listen normally on 5 lo ::1 UDP 123
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: Listen normally on 6 eno1 fe80::225:b5ff:fe00:67bf UDP 123
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: Listening on routing socket on fd #23 for interface updates
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c016 06 restart
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c012 02 freq_set kernel 0.000 PPM
Nov 20 23:27:25 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c011 01 freq_not_set
Nov 20 23:27:26 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c61c 0c clock_step -85043.979371 s
Nov 19 23:50:02 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c614 04 freq_mode
Nov 19 23:50:03 DDC-DTR-2.cisco.com ntpd[17377]: 0.0.0.0 c618 08 no_sys_peer
```

7. Querying the NTP source to check on the clock-skew if any:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/ntpdate -q 171.68.38.66"
UCP-Ctrl-2 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003010, delay 0.24805
19 Nov 23:52:47 ntpdate[27036]: adjust time server 171.68.38.66 offset 0.003010 sec

DDC-DTR-1 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003203, delay 0.24835
19 Nov 23:52:47 ntpdate[17657]: adjust time server 171.68.38.66 offset 0.003203 sec

DDC-DTR-2 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003155, delay 0.24808
19 Nov 23:52:47 ntpdate[17685]: adjust time server 171.68.38.66 offset 0.003155 sec

UCP-Ctrl-1 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.002539, delay 0.24806
19 Nov 23:52:47 ntpdate[29624]: adjust time server 171.68.38.66 offset 0.002539 sec

UCP-Ctrl-3 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003849, delay 0.24835
19 Nov 23:52:47 ntpdate[26796]: adjust time server 171.68.38.66 offset 0.003849 sec

UCP-Node-1 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003511, delay 0.24796
19 Nov 23:52:55 ntpdate[26610]: adjust time server 171.68.38.66 offset 0.003511 sec

DDC-DTR-3 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003190, delay 0.24765
19 Nov 23:52:55 ntpdate[17588]: adjust time server 171.68.38.66 offset 0.003190 sec

UCP-Node-3 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.001835, delay 0.24794
19 Nov 23:52:55 ntpdate[26274]: adjust time server 171.68.38.66 offset 0.001835 sec

UCP-Node-2 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.003033, delay 0.24799
19 Nov 23:52:55 ntpdate[26812]: adjust time server 171.68.38.66 offset 0.003033 sec

UCP-Node-4 | SUCCESS | rc=0 >>
server 171.68.38.66, stratum 1, offset 0.002664, delay 0.24838
19 Nov 13:22:55 ntpdate[27417]: adjust time server 171.68.38.66 offset 0.002664 sec
```

Installing Cisco Virtual Interface Card (VIC) eNIC (Ethernet Network Interface Card) Driver

RHEL 7.2 comes with inbox eNIC driver for the Cisco VIC, we need to update it to the latest version for the UCS Manager release used in the solution. RPM needs to be extracted from downloaded Cisco UCS B-Series Server Software bundle from cisco.com.

1. Installing of eNIC driver can be done through rpm install and this requires host reboot.
2. Copy the rpm file to all the nodes using Ansible:

```
[root@UCP-Ctrl-1 ~]# ls -ltr
total 716
-rw-r--r--. 1 root root 705956 Sep 14 04:57 kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm
-rw-------. 1 root root   1852 Nov 18 12:46 anaconda-ks.cfg
-rw-------. 1 root root   1945 Nov 18 13:34 initial-setup-ks.cfg
-rw-r--r--. 1 root root  14612 Nov 19 22:51 epel-release-latest-7.noarch.rpm
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm dest=/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "21ee851c25a323feeafde836af00b326312df47b",
    "dest": "/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm",
    "secontext": "unconfined_u:object_r:admin_home_t:s0",
    "size": 705956,
    "state": "file",
    "uid": 0
}
DDC-DTR-2 | SUCCESS => {
    "changed": true,
    "checksum": "21ee851c25a323feeafde836af00b326312df47b",
    "dest": "/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm",
    "gid": 0,
    "group": "root",
    "md5sum": "fcd25b89826487bab805850dfde9e690",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:admin_home_t:s0",
    "size": 705956,
    "src": "/root/.ansible/tmp/ansible-tmp-1479580441.19-220251164171267/source",
    "state": "file",
    "uid": 0
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": true,
    "checksum": "21ee851c25a323feeafde836af00b326312df47b",
    "dest": "/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm",
    "gid": 0,
    "group": "root",
    "md5sum": "fcd25b89826487bab805850dfde9e690",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:admin_home_t:s0",
    "size": 705956,
    "src": "/root/.ansible/tmp/ansible-tmp-1479580441.19-229072103837751/source",
    "state": "file",
    "uid": 0
}
```

3. Use the yum module to install the eNIC driver rpm file on all node through Ansible:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m yum -a "name=/root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm state=present"
DDC-DTR-1 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nExamining /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm: kmod-enic-2.3
.0.30-rhel7u2.el7.x86_64\nMarking /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm to be installed\nResolving Dependencies\n--> Running transaction check\n---> Package kmod-enic.x86_64
 0:2.3.0.30-rhel7u2.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
===\n Package\n       Arch    Version              Repository                          Size\n================================================================================\nInsta
lling:\n kmod-enic\n       x86_64 2.3.0.30-rhel7u2.el7 /kmod-enic-2.3.0.30-rhel7u2.el7.x86_64 3.8 M\n\nTransaction Summary\n================================================================================
==========\nInstall  1 Package\n\nTotal size: 3.8 M\nInstalled size: 3.8 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test suc
ceeded\nRunning transaction\n  Installing : kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                                  1/1 \n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change loca
le (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale:
 LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n  Verifying  :
 kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                     1/1 \n\nInstalled:\n  kmod-enic.x86_64 0:2.3.0.30-rhel7u2.el7                                                \n\nComplete!\n"
    ]
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nExamining /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm: kmod-enic-2.3
.0.30-rhel7u2.el7.x86_64\nMarking /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm to be installed\nResolving Dependencies\n--> Running transaction check\n---> Package kmod-enic.x86_64
 0:2.3.0.30-rhel7u2.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
===\n Package\n       Arch    Version              Repository                          Size\n================================================================================\nInsta
lling:\n kmod-enic\n       x86_64 2.3.0.30-rhel7u2.el7 /kmod-enic-2.3.0.30-rhel7u2.el7.x86_64 3.8 M\n\nTransaction Summary\n================================================================================
==========\nInstall  1 Package\n\nTotal size: 3.8 M\nInstalled size: 3.8 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test suc
ceeded\nRunning transaction\n  Installing : kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                                  1/1 \n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change loca
le (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale:
 LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n  Verifying  :
 kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                     1/1 \n\nInstalled:\n  kmod-enic.x86_64 0:2.3.0.30-rhel7u2.el7                                                \n\nComplete!\n"
    ]
}
UCP-Ctrl-1 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-\n              : manager\nExamining /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm: kmod-enic-2.3
.0.30-rhel7u2.el7.x86_64\nMarking /root/kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm to be installed\nResolving Dependencies\n--> Running transaction check\n---> Package kmod-enic.x86_64
 0:2.3.0.30-rhel7u2.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================
===\n Package\n       Arch    Version              Repository                          Size\n================================================================================\nInsta
lling:\n kmod-enic\n       x86_64 2.3.0.30-rhel7u2.el7 /kmod-enic-2.3.0.30-rhel7u2.el7.x86_64 3.8 M\n\nTransaction Summary\n================================================================================
==========\nInstall  1 Package\n\nTotal size: 3.8 M\nInstalled size: 3.8 M\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test suc
ceeded\nRunning transaction\n  Installing : kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                                  1/1 \n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change loca
le (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale:
 LC_CTYPE: cannot change locale (): No such file or directory\n/sbin/dracut: line 640: warning: setlocale: LC_CTYPE: cannot change locale (): No such file or directory\n  Verifying  :
 kmod-enic-2.3.0.30-rhel7u2.el7.x86_64                     1/1 \n\nInstalled:\n  kmod-enic.x86_64 0:2.3.0.30-rhel7u2.el7                                                \n\nComplete!\n"
    ]
}
```

4. Reboot all the nodes after installing the latest eNIC driver:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/shutdown -r +1"

Broadcast message from root@UCP-Ctrl-1.cisco.com (Sun 2016-11-20 00:21:35 IST):

The system is going down for reboot at Sun 2016-11-20 00:22:35 IST!

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@DDC-DTR-1.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot at
Sun 2016-11-20 00:22:35 IST!

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@DDC-DTR-2.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot at
Sun 2016-11-20 00:22:35 IST!

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@UCP-Ctrl-2.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot
at Sun 2016-11-20 00:22:35 IST!

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@UCP-Ctrl-3.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot
at Sun 2016-11-20 00:22:35 IST!

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

DDC-DTR-1 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

DDC-DTR-2 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@DDC-DTR-3.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot at
Sun 2016-11-20 00:22:35 IST!

DDC-DTR-3 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

 [WARNING]: Module invocation had junk after the JSON data:   Broadcast message from root@UCP-Node-1.cisco.com (Sun 2016-11-20 00:21:35 IST):    The system is going down for reboot
at Sun 2016-11-20 00:22:35 IST!

UCP-Node-1 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.

UCP-Node-2 | SUCCESS | rc=0 >>
Shutdown scheduled for Sun 2016-11-20 00:22:35 IST, use 'shutdown -c' to cancel.
```

Configuring Host OS Firewall for required ports

Docker Datacenter requires some TCP and UDP ports to be opened to facilitate communication between its container infrastructure services running on cluster nodes. This needs to be done before installing Docker CS Engine and the Docker UCP. For opening ports on hosts, 'firewall-cmd' is used for configuring 'firewalld-service' as shown below. For every port type such as TCP and UDP, refer the table below to see the specific ports to be opened and then making them permanent on the host.

Table 1. TCP and UDP ports to be opened on hosts for Docker UCP

Hosts	Direction	Port	Purpose
controllers, nodes	in	TCP 443 (configurable)	Web app and CLI client access to UCP.
controllers, nodes	in	TCP 2375	Heartbeat for nodes, to ensure they are running.
controllers	in	TCP 2376 (configurable)	Swarm manager accepts requests from UCP controller.
controllers, nodes	in, out	TCP + UDP 4789	Overlay networking.
controllers, nodes	in, out	TCP + UDP 7946	Overlay networking.
controllers, nodes	in	TCP 12376	Proxy for TLS, provides access to UCP, Swarm, and Engine.
controller	in	TCP 12379	Internal node configuration, cluster configuration, and HA.
controller	in	TCP 12380	Internal node configuration, cluster configuration, and HA.
controller	in	TCP 12381	Proxy for TLS, provides access to UCP.
controller	in	TCP 12382	Manages TLS and requests from swarm manager.
controller	in	TCP 12383	Used by the authentication storage backend.
controller	in	TCP 12384	Used by authentication storage backend for replication across controllers.
controller	in	TCP 12385	The port where the authentication API is exposed.
controller	in	TCP 12386	Used by the authentication worker.

For second architecture, since UCP and DTR services are co-located, we need to open port 4443 on all the UCP Controller nodes.

The screenshot below shows the command to open the TCP 443 port using firewall-cmd:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=443/tcp"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

Making the opened ports permanent:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=443/tcp --permanent"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

For a range of consecutive ports, we can use the command shown below to open the successive ports 2375 to 2376:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=2375-2376/tcp"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=2375-2376/tcp --permanent"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

Following is an example for opening 4789 UDP port and making the port permanent:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=4789/udp"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=4789/udp --permanent"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success
```

Repeat the commands for the rest of the ports on all the nodes and complete opening the firewall ports. Once this task is complete, restart ‘firewalld-service’ as below:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl restart firewalld.service"
UCP-Ctrl-3 | SUCCESS | rc=0 >>


DDC-DTR-1 | SUCCESS | rc=0 >>


UCP-Ctrl-2 | SUCCESS | rc=0 >>


DDC-DTR-2 | SUCCESS | rc=0 >>


UCP-Ctrl-1 | SUCCESS | rc=0 >>


DDC-DTR-3 | SUCCESS | rc=0 >>


UCP-Node-2 | SUCCESS | rc=0 >>


UCP-Node-3 | SUCCESS | rc=0 >>


UCP-Node-1 | SUCCESS | rc=0 >>


UCP-Node-4 | SUCCESS | rc=0 >>
```

To confirm the list of ports opened run the following command as shown below:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --list-all"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
public (default, active)
  interfaces: eno1
  sources:
  services: dhcpv6-client ssh
  ports: 443/tcp 4789/udp 7946/udp 7946/tcp 4789/tcp 12379-12386/tcp 2375-2376/tcp 12376/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:


DDC-DTR-2 | SUCCESS | rc=0 >>
public (default, active)
  interfaces: eno1
  sources:
  services: dhcpv6-client ssh
  ports: 443/tcp 4789/udp 7946/udp 7946/tcp 4789/tcp 12379-12386/tcp 2375-2376/tcp 12376/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:


UCP-Ctrl-2 | SUCCESS | rc=0 >>
public (default, active)
  interfaces: eno1
  sources:
  services: dhcpv6-client ssh
  ports: 443/tcp 4789/udp 7946/udp 7946/tcp 4789/tcp 12379-12386/tcp 2375-2376/tcp 12376/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:


DDC-DTR-1 | SUCCESS | rc=0 >>
public (default, active)
  interfaces: eno1
  sources:
  services: dhcpv6-client ssh
  ports: 443/tcp 4789/udp 7946/udp 7946/tcp 4789/tcp 12379-12386/tcp 2375-2376/tcp 12376/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
```

Installation of Docker Repo and Engine

Before we proceed to install the Docker repo and Docker Engine we need to make sure we have removed the epel repos from all the nodes. This is needed as these repos are not required anymore and can possibly create issue during other software installations.

1. To remove epel repo from all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/rpm -evh epel-release-7-8.noarch"
UCP-Ctrl-2 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

DDC-DTR-2 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

DDC-DTR-1 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Node-2 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

DDC-DTR-3 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Node-3 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Node-4 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################

UCP-Node-1 | SUCCESS | rc=0 >>
Preparing...                          ########################################
Cleaning up / removing...
epel-release-7-8                      ########################################
```

2. Do the yum database cleanup:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/yum clean all"
DDC-DTR-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything

DDC-DTR-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything

DDC-DTR-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Cleaning repos: packages.docker.com_1.12_yum_repo_main_centos_7
              : rhel-7-server-extras-rpms rhel-7-server-optional-rpms
              : rhel-7-server-rpms
Cleaning up everything
```

3. Install ‘yum-utils’ package:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m yum -a "name=yum-utils state=present"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
DDC-DTR-2 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
DDC-DTR-1 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
DDC-DTR-3 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "yum-utils-1.1.31-34.el7.noarch providing yum-utils is already installed"
    ]
}
```

4. Add the Docker public key for Docker CS Engine packages on all the hosts:

```
[root@UCP-Ctrl-2 ~]# rpm --import "https://sks-keyservers.net/pks/lookup?op=get&search=0xee6d536cf7dc86e2d7d56f59a178ac6c6238f52e"
[root@UCP-Ctrl-2 ~]#
```

5. Add Docker repository on all the hosts:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/yum-config-manager --add-repo https://packages.docker.com/1.12/yum/repo/main/centos/7"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7

[packages.docker.com_1.12_yum_repo_main_centos_7]
name=added from: https://packages.docker.com/1.12/yum/repo/main/centos/7
baseurl=https://packages.docker.com/1.12/yum/repo/main/centos/7
enabled=1

DDC-DTR-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7

[packages.docker.com_1.12_yum_repo_main_centos_7]
name=added from: https://packages.docker.com/1.12/yum/repo/main/centos/7
baseurl=https://packages.docker.com/1.12/yum/repo/main/centos/7
enabled=1

DDC-DTR-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7

[packages.docker.com_1.12_yum_repo_main_centos_7]
name=added from: https://packages.docker.com/1.12/yum/repo/main/centos/7
baseurl=https://packages.docker.com/1.12/yum/repo/main/centos/7
enabled=1

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7

[packages.docker.com_1.12_yum_repo_main_centos_7]
name=added from: https://packages.docker.com/1.12/yum/repo/main/centos/7
baseurl=https://packages.docker.com/1.12/yum/repo/main/centos/7
enabled=1

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7

[packages.docker.com_1.12_yum_repo_main_centos_7]
name=added from: https://packages.docker.com/1.12/yum/repo/main/centos/7
baseurl=https://packages.docker.com/1.12/yum/repo/main/centos/7
enabled=1

DDC-DTR-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id
adding repo from: https://packages.docker.com/1.12/yum/repo/main/centos/7
```

6. Execute ‘yum repolist’ command to refresh the host software repos:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/yum repolist"
UCP-Ctrl-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
repo id                                          repo name                status
packages.docker.com_1.12_yum_repo_main_centos_7 added from: https://packa     20
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux     327
rhel-7-server-optional-rpms/x86_64               Red Hat Enterprise Linux   8,903
rhel-7-server-rpms/x86_64                        Red Hat Enterprise Linux  11,412
repolist: 20,662

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
repo id                                          repo name                status
packages.docker.com_1.12_yum_repo_main_centos_7 added from: https://packa     20
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux     327
rhel-7-server-optional-rpms/x86_64               Red Hat Enterprise Linux   8,903
rhel-7-server-rpms/x86_64                        Red Hat Enterprise Linux  11,412
repolist: 20,662

DDC-DTR-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
repo id                                          repo name                status
packages.docker.com_1.12_yum_repo_main_centos_7 added from: https://packa     20
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux     327
rhel-7-server-optional-rpms/x86_64               Red Hat Enterprise Linux   8,903
rhel-7-server-rpms/x86_64                        Red Hat Enterprise Linux  11,412
repolist: 20,662

UCP-Ctrl-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
repo id                                          repo name                status
packages.docker.com_1.12_yum_repo_main_centos_7 added from: https://packa     20
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux     327
rhel-7-server-optional-rpms/x86_64               Red Hat Enterprise Linux   8,903
rhel-7-server-rpms/x86_64                        Red Hat Enterprise Linux  11,412
repolist: 20,662

DDC-DTR-1 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
repo id                                          repo name                status
packages.docker.com_1.12_yum_repo_main_centos_7 added from: https://packa     20
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux     327
rhel-7-server-optional-rpms/x86_64               Red Hat Enterprise Linux   8,903
rhel-7-server-rpms/x86_64                        Red Hat Enterprise Linux  11,412
repolist: 20,662
```

7. Install Docker Engine on all the nodes using Ansible:

8. Verify that all the nodes have Docker Engine installed successfully:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/yum info docker-engine"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Installed Packages
Name        : docker-engine
Arch        : x86_64
Version     : 1.12.3.cs4
Release     : 1.el7.centos
Size        : 79 M
Repo        : installed
From repo   : packages.docker.com_1.12_yum_repo_main_centos_7
Summary     : The open-source application container engine
URL         : https://dockerproject.org
License     : ASL 2.0
Description : Docker is an open source project to build, ship and run any
            : application as a lightweight container.
            :
            : Docker containers are both hardware-agnostic and
            : platform-agnostic. This means they can run anywhere, from your
            : laptop to the largest EC2 compute instance and everything in
            : between - and they don't require you to use a particular language,
            : framework or packaging system. That makes them great building
            : blocks for deploying and scaling web apps, databases, and backend
            : services without depending on a particular stack or provider.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
Installed Packages
Name        : docker-engine
Arch        : x86_64
Version     : 1.12.3.cs4
Release     : 1.el7.centos
Size        : 79 M
Repo        : installed
From repo   : packages.docker.com_1.12_yum_repo_main_centos_7
Summary     : The open-source application container engine
URL         : https://dockerproject.org
License     : ASL 2.0
Description : Docker is an open source project to build, ship and run any
            : application as a lightweight container.
            :
            : Docker containers are both hardware-agnostic and
            : platform-agnostic. This means they can run anywhere, from your
            : laptop to the largest EC2 compute instance and everything in
            : between - and they don't require you to use a particular language,
            : framework or packaging system. That makes them great building
            : blocks for deploying and scaling web apps, databases, and backend
            : services without depending on a particular stack or provider.
```

Do not start the Docker daemon services as yet. We need to configure Docker device-mapper driver in direct LVM-Mode before starting the Docker daemon services.

Configuring Docker CS Engine for Device-Mapper Driver in Direct LVM-Mode

Device Mapper is a kernel-based framework that underpins many advanced volume management technologies on Linux. Docker's device-mapper storage driver leverages the thin provisioning and snapshotting capabilities of this framework for image and container management. The preferred configuration for production deployments is direct-lvm. This mode uses block devices to create the thin pool. The following procedure shows you how to configure a Docker host to use the device-mapper storage driver in a direct-lvm configuration.

We will be using /dev/sdb device on each node for creating thin pool for Docker use. Each node has 200 GB size lun getting discovered as /dev/sdb, through Cisco UCS Manager storage profile configuration in RAID-1 level.

1. Create physical volume on /dev/sdb on all the hosts:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/pvcreate /dev/sdb"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

DDC-DTR-2 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

DDC-DTR-1 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

DDC-DTR-3 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Node-2 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Node-3 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Node-1 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created

UCP-Node-4 | SUCCESS | rc=0 >>
  Physical volume "/dev/sdb" successfully created
```

2. Verify that the physical volume gets created cleanly:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/pvs -a"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
  PV               VG      Fmt  Attr PSize    PFree
  /dev/rhel/home                --- 0         0
  /dev/rhel/root                --- 0         0
  /dev/rhel/swap                --- 0         0
  /dev/sda1                     --- 0         0
  /dev/sda2        rhel    lvm2 a--  59.51g  60.00m
  /dev/sdb         Docker  lvm2 a--  200.00g 200.00g

DDC-DTR-1 | SUCCESS | rc=0 >>
  PV               VG      Fmt  Attr PSize    PFree
  /dev/rhel/home                --- 0         0
  /dev/rhel/root                --- 0         0
  /dev/rhel/swap                --- 0         0
  /dev/sda1                     --- 0         0
  /dev/sda2        rhel    lvm2 a--  59.51g  60.00m
  /dev/sdb         Docker  lvm2 a--  200.00g 200.00g

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  PV               VG      Fmt  Attr PSize    PFree
  /dev/rhel/home                --- 0         0
  /dev/rhel/root                --- 0         0
  /dev/rhel/swap                --- 0         0
  /dev/sda1                     --- 0         0
  /dev/sda2        rhel    lvm2 a--  59.51g  60.00m
  /dev/sdb         Docker  lvm2 a--  200.00g 200.00g

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  PV               VG      Fmt  Attr PSize    PFree
  /dev/rhel/home                --- 0         0
  /dev/rhel/root                --- 0         0
  /dev/rhel/swap                --- 0         0
  /dev/sda1                     --- 0         0
  /dev/sda2        rhel    lvm2 a--  59.51g  60.00m
  /dev/sdb         Docker  lvm2 a--  200.00g 200.00g

DDC-DTR-2 | SUCCESS | rc=0 >>
  PV               VG      Fmt  Attr PSize    PFree
  /dev/rhel/home                --- 0         0
  /dev/rhel/root                --- 0         0
  /dev/rhel/swap                --- 0         0
  /dev/sda1                     --- 0         0
  /dev/sda2        rhel    lvm2 a--  59.51g  60.00m
  /dev/sdb         Docker  lvm2 a--  200.00g 200.00g
```

3. Create a volume group named Docker using physical volume which we previously created:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/vgcreate Docker /dev/sdb"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

DDC-DTR-2 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

DDC-DTR-1 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

DDC-DTR-3 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Node-2 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Node-3 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Node-4 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created

UCP-Node-1 | SUCCESS | rc=0 >>
  Volume group "Docker" successfully created
```

4. Create a thin pool named thinpool. In this example, the logical data is 95% of the ‘Docker’ volume group size. Leaving 5% free space allows for auto expanding of either the data or metadata if space runs low.

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/lvcreate --wipesignatures y -n thinpool Docker -l 95%VG"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

DDC-DTR-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

DDC-DTR-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Node-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

DDC-DTR-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Node-4 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Node-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.

UCP-Node-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" created.
```

5. Convert the pool into thinpool:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/lvcreate --wipesignatures y -n thinpoolmeta Docker -l 1%VG"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

DDC-DTR-1 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

DDC-DTR-2 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

DDC-DTR-3 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Node-2 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Node-1 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Node-4 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.

UCP-Node-3 | SUCCESS | rc=0 >>
  Logical volume "thinpoolmeta" created.
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/lvconvert -y --zero n -c 512K --thinpool Docker/thinpool --poolmetadata Docker/thinpoolmeta"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

DDC-DTR-1 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

DDC-DTR-2 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Node-2 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

DDC-DTR-3 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Node-3 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Node-4 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

UCP-Node-1 | SUCCESS | rc=0 >>
  Converted Docker/thinpool to thin pool.  WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to pool's data and metadata volumes.
  THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)
```

6. Configure auto-execution of thin pool via lvm profile –

```
[root@UCP-Ctrl-1 ~]# #vi /etc/lvm/profile/Docker-thinpool.profile
[root@UCP-Ctrl-1 ~]# cat /etc/lvm/profile/Docker-thinpool.profile
activation {
    thin_pool_autoextend_threshold=80
    thin_pool_autoextend_percent=20
}
(reverse-i-search)`ansible Docker -': ^Csible Docker -a "/usr/sbin/lvconvert -y --zero n -c 512K --thinpool Docker/thinpool --poolmetadata Docker/thinpoolmeta"
(reverse-i-search)`-m': ansible Docker ^C yum -a "name=docker-engine state=present"
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/^Cest=/etc/environment"
[root@UCP-Ctrl-1 ~]# clear
[root@UCP-Ctrl-1 ~]# #vi /etc/lvm/profile/Docker-thinpool.profile
[root@UCP-Ctrl-1 ~]# cat /etc/lvm/profile/Docker-thinpool.profile
activation {
    thin_pool_autoextend_threshold=80
    thin_pool_autoextend_percent=20
}
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/etc/lvm/profile/Docker-thinpool.profile dest=//etc/lvm/profile/Docker-thinpool.profile"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "7ddbe4eb9f7dcc60fb6cfcf4c0098329cdad0594",
    "dest": "//etc/lvm/profile/Docker-thinpool.profile",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "//etc/lvm/profile/Docker-thinpool.profile",
    "secontext": "unconfined_u:object_r:lvm_etc_t:s0",
    "size": 89,
    "state": "file",
    "uid": 0
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": true,
    "checksum": "7ddbe4eb9f7dcc60fb6cfcf4c0098329cdad0594",
    "dest": "//etc/lvm/profile/Docker-thinpool.profile",
    "gid": 0,
    "group": "root",
    "md5sum": "bb00a1e54c6a3bd0703c584af9387dca",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:lvm_etc_t:s0",
    "size": 89,
    "src": "/root/.ansible/tmp/ansible-tmp-1479803534.77-114938113137518/source",
    "state": "file",
    "uid": 0
}
```

We have created Docker-thinpool profile by setting auto-extend threshold to 80% and auto-extend space to 20%. These values are set to make sure that we get 80% of the total available space at one go and later when the utilization reaches 80% fully; we would get 20% of the remaining extended space.

7. Apply newly created lvm-profile:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/lvchange --metadataprofile Docker-thinpool Docker/thinpool"
DDC-DTR-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

DDC-DTR-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Ctrl-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Node-2 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

DDC-DTR-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Node-1 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Node-3 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.

UCP-Node-4 | SUCCESS | rc=0 >>
  Logical volume "thinpool" changed.
```

8. Setup LV monitoring:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/sbin/lvs -o+seg_monitor"
DDC-DTR-1 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

UCP-Ctrl-2 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

UCP-Ctrl-3 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

DDC-DTR-2 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

UCP-Ctrl-1 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

UCP-Node-2 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g

UCP-Node-1 | SUCCESS | rc=0 >>
  LV       VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert Monitor
  thinpool Docker twi-a-t--- 190.00g             0.00   0.01                             monitored
  home     rhel   -wi-ao----  18.19g
  root     rhel   -wi-ao----  37.26g
  swap     rhel   -wi-ao----   4.00g
```

9. Configure Docker daemon with specific device-mapper options. Now that storage is configured we need to tell Docker daemon to use it. We do this by creating daemon.jason file with the following configuration parameters and place it as /etc/docker/daemon.json:

```
[root@UCP-Ctrl-1 ~]# #vi /etc/docker/daemon.json
[root@UCP-Ctrl-1 ~]# cat /etc/docker/daemon.json
{
  "storage-driver": "devicemapper",
   "storage-opts": [
     "dm.thinpooldev=/dev/mapper/Docker-thinpool",
     "dm.use_deferred_removal=true",
     "dm.use_deferred_deletion=true"
   ]
}
```

9. If /etc/docker does not exist, create it before-hand on all the nodes and copy the daemon.json:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/mkdir /etc/docker"
UCP-Ctrl-3 | SUCCESS | rc=0 >>


DDC-DTR-1 | SUCCESS | rc=0 >>


UCP-Ctrl-1 | FAILED | rc=1 >>
/usr/bin/mkdir: cannot create directory `/etc/docker': File exists

DDC-DTR-2 | SUCCESS | rc=0 >>


UCP-Ctrl-2 | SUCCESS | rc=0 >>


DDC-DTR-3 | SUCCESS | rc=0 >>


UCP-Node-1 | SUCCESS | rc=0 >>


UCP-Node-3 | SUCCESS | rc=0 >>


UCP-Node-2 | SUCCESS | rc=0 >>


UCP-Node-4 | SUCCESS | rc=0 >>
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/etc/docker/daemon.json dest=/etc/docker/daemon.json"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "eaac4b28dcefa7d4326e8db898d4452a2812f6e5",
    "dest": "/etc/docker/daemon.json",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/docker/daemon.json",
    "secontext": "unconfined_u:object_r:etc_t:s0",
    "size": 191,
    "state": "file",
    "uid": 0
}
DDC-DTR-2 | SUCCESS => {
    "changed": true,
    "checksum": "eaac4b28dcefa7d4326e8db898d4452a2812f6e5",
    "dest": "/etc/docker/daemon.json",
    "gid": 0,
    "group": "root",
    "md5sum": "b796711c7ba52654a7b5d8e740a231d4",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:docker_config_t:s0",
    "size": 191,
    "src": "/root/.ansible/tmp/ansible-tmp-1479804595.72-176691544304356/source",
    "state": "file",
    "uid": 0
}
UCP-Ctrl-2 | SUCCESS => {
    "changed": true,
    "checksum": "eaac4b28dcefa7d4326e8db898d4452a2812f6e5",
    "dest": "/etc/docker/daemon.json",
    "gid": 0,
    "group": "root",
    "md5sum": "b796711c7ba52654a7b5d8e740a231d4",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:docker_config_t:s0",
    "size": 191,
    "src": "/root/.ansible/tmp/ansible-tmp-1479804595.73-272322441151302/source",
    "state": "file",
    "uid": 0
}
```

10. Enable Docker service and start the service on all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl enable docker.service"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Ctrl-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

DDC-DTR-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

DDC-DTR-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Ctrl-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

DDC-DTR-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Node-3 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Node-1 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Node-2 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

UCP-Node-4 | SUCCESS | rc=0 >>
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl start docker.service"
UCP-Ctrl-2 | SUCCESS | rc=0 >>

UCP-Ctrl-3 | SUCCESS | rc=0 >>

UCP-Ctrl-1 | SUCCESS | rc=0 >>

DDC-DTR-2 | SUCCESS | rc=0 >>

DDC-DTR-1 | SUCCESS | rc=0 >>

UCP-Node-2 | SUCCESS | rc=0 >>

DDC-DTR-3 | SUCCESS | rc=0 >>

UCP-Node-1 | SUCCESS | rc=0 >>

UCP-Node-3 | SUCCESS | rc=0 >>

UCP-Node-4 | SUCCESS | rc=0 >>
```

11. Verify that Docker service is up and running on all the nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl status docker.service"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2016-11-22 14:21:43 IST; 40s ago
     Docs: https://docs.docker.com
 Main PID: 26108 (dockerd)
   Memory: 34.1M
   CGroup: /system.slice/docker.service
           ├─26108 /usr/bin/dockerd
           └─26114 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --shim docker-containerd-shim --metrics-interval=0 --start-timeout 2m --state-dir /
var/run/docker/libcontainerd/containerd --runtime docker-runc

Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.365762369+05:30" level=info msg="Graph migration to content-addressability took 0.00 seconds"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.366249228+05:30" level=warning msg="mountpoint for pids not found"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.366465234+05:30" level=info msg="Loading containers: start."
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.374960029+05:30" level=info msg="Firewalld running: true"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.579016146+05:30" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16.
Daemon option --bip can be used to set a preferred IP address"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.734442948+05:30" level=info msg="Loading containers: done."
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.734826894+05:30" level=info msg="Daemon has completed initialization"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.734823034+05:30" level=info msg="Docker daemon" commit=65c6c4c graphdriver=devicemapper version=1.12.3-c
s4
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com dockerd[26108]: time="2016-11-22T14:21:43.746362287+05:30" level=info msg="API listen on /var/run/docker.sock"
Nov 22 14:21:43 UCP-Ctrl-3.cisco.com systemd[1]: Started Docker Application Container Engine.

DDC-DTR-2 | SUCCESS | rc=0 >>
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2016-11-22 14:21:43 IST; 40s ago
     Docs: https://docs.docker.com
 Main PID: 26003 (dockerd)
   Memory: 32.8M
   CGroup: /system.slice/docker.service
           ├─26003 /usr/bin/dockerd
           └─26009 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --shim docker-containerd-shim --metrics-interval=0 --start-timeout 2m --state-dir /
var/run/docker/libcontainerd/containerd --runtime docker-runc

Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.610670699+05:30" level=info msg="Graph migration to content-addressability took 0.00 seconds"
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.611038026+05:30" level=warning msg="mountpoint for pids not found"
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.611316553+05:30" level=info msg="Loading containers: start."
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.620706107+05:30" level=info msg="Firewalld running: true"
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.617210466+05:30" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. D
aemon option --bip can be used to set a preferred IP address"
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.772893703+05:30" level=info msg="Loading containers: done."
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.773105655+05:30" level=info msg="Daemon has completed initialization"
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.773122744+05:30" level=info msg="Docker daemon" commit=65c6c4c graphdriver=devicemapper version=1.12.3-cs
4
Nov 22 14:21:43 DDC-DTR-2.cisco.com dockerd[26003]: time="2016-11-22T14:21:43.784878321+05:30" level=info msg="API listen on /var/run/docker.sock"
Nov 22 14:21:43 DDC-DTR-2.cisco.com systemd[1]: Started Docker Application Container Engine.
```

12. Verify that the Docker service is running with storage driver set to device-mapper:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/docker info"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.12.3-cs4
Storage Driver: devicemapper
 Pool Name: Docker-thinpool
 Pool Blocksize: 524.3 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
 Data file:
 Metadata file:
 Data Space Used: 19.92 MB
 Data Space Total: 204 GB
 Data Space Available: 204 GB
 Metadata Space Used: 266.2 kB
 Metadata Space Total: 2.143 GB
 Metadata Space Available: 2.143 GB
 Thin Pool Minimum Free Space: 20.4 GB
 Udev Sync Supported: true
 Deferred Removal Enabled: true
 Deferred Deletion Enabled: true
 Deferred Deleted Device Count: 0
 Library Version: 1.02.107-RHEL7 (2016-06-09)
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: null overlay host bridge
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Security Options: seccomp
Kernel Version: 3.10.0-327.36.3.el7.x86_64
Operating System: Red Hat Enterprise Linux
OSType: linux
Architecture: x86_64
CPUs: 56
Total Memory: 125.5 GiB
Name: UCP-Ctrl-3.cisco.com
ID: KU2O:W4OU:TGHX:DGU7:7A2I:J44S:J7ZT:D7LD:HSBI:HCS3:BFEO:HFMV
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Insecure Registries:
 127.0.0.0/8WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
```

13. Check that the LVs are monitored as configured:

```
[root@UCP-Ctrl-1 ~]# journalctl -fu dm-event.service
-- Logs begin at Tue 2016-11-22 00:05:59 IST. --
Nov 22 13:57:22 UCP-Ctrl-1.cisco.com systemd[1]: Started Device-mapper event daemon.
Nov 22 13:57:22 UCP-Ctrl-1.cisco.com systemd[1]: Starting Device-mapper event daemon...
Nov 22 13:57:22 UCP-Ctrl-1.cisco.com dmeventd[26867]: dmeventd ready for processing.
Nov 22 13:57:22 UCP-Ctrl-1.cisco.com lvm[26867]: Monitoring thin Docker-thinpool.
^C
```

14. Check to see there aren’t any containers running on any of the cluster nodes:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/docker ps"
DDC-DTR-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Ctrl-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

DDC-DTR-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Ctrl-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Ctrl-3 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

DDC-DTR-3 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Node-3 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Node-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Node-4 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

UCP-Node-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

With this all the cluster nodes are installed with Docker CS Engine and they are configured for installing Docker UCP and DTR services.

This brings us to a state where all cluster nodes are installed with Docker CS Engine and they are configured for installing Docker UCP and DTR services.

Install and Configure Docker UCP Controller Nodes

Docker UCP is a containerized application that requires Docker CS Engine 1.10.0 or above to run. Installation work flow is split into two steps. We identify the first UCP Controller node, install UCP on it and then start adding UCP Controller replicas and UCP Nodes. This way they form a large cluster running Docker Datacenter. Docker Trusted Registry is again a containerized application running on one of the UCP nodes.

1. Docker UCP installation is one single command as shown below:

```
[root@UCP-Ctrl-1 ~]# docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:1.1.4 install -i --host-address 10.65.122.61
Unable to find image 'docker/ucp:1.1.4' locally
1.1.4: Pulling from docker/ucp
c0cb142e4345: Pull complete
bcee58a6e550: Pull complete
dbd82e351077: Pull complete
Digest: sha256:d16275fabledac53cd19e7b408581fb3a115013c721fabbc1e4f813abe272065
Status: Downloaded newer image for docker/ucp:1.1.4
INFO[0000] Verifying your system is compatible with UCP
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
WARN[0000] Your system uses devicemapper.  We can not accurately detect available storage space.  Please make sure you have at least 3.00 GB available in /var/lib/docker
Please choose your initial UCP admin password:
Confirm your initial password:
ERRO[0017] Passwords don't match, please try again
Please choose your initial UCP admin password:
Confirm your initial password:
INFO[0026] Pulling required images... (this may take a while)
We detected the following hostnames/IP addresses for this system [UCP-Ctrl-1.cisco.com 127.0.0.1 172.17.0.1 10.65.122.61]

You may enter additional aliases (SANs) now or press enter to proceed with the above list.
Additional aliases:
INFO[1910] Installing UCP with host address 10.65.122.61 - If this is incorrect, please specify an alternative address with the '--host-address' flag
INFO[0000] Checking that required ports are available and accessible
INFO[0014] Generating UCP Cluster Root CA
INFO[0033] Generating UCP Client Root CA
INFO[0038] Deploying UCP Containers
WARN[0097] Configuration updated. You will have to manually restart the docker daemon for the changes to take effect.
INFO[0097] UCP instance ID: RMLG:3QSI:LSXS:Z4MD:PBEZ:KYAS:5PRZ:EUAJ:JP3F:R3XO:OABM:HPUI
INFO[0097] UCP Server SSL: SHA-256 Fingerprint=69:60:08:36:96:5C:01:E5:0A:A9:19:4C:7D:1B:A4:CF:E6:BB:24:FF:E3:8C:FF:12:65:C1:83:53:55:11:29:E8
INFO[0097] Login as "admin"/(your admin password) to UCP at https://10.65.122.61:443
```

This command takes the following parameters:

- Container name: --name ucp
- UCP version tag: docker/ucp:1.1.4 *** this is important as we are specifically saying what version needed to be installed. Otherwise ‘docker run’ command will download and install the latest UCP version.

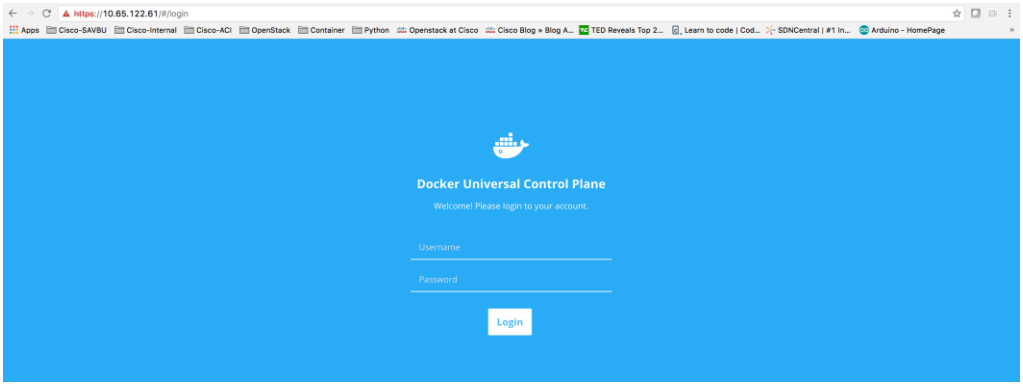
- UCP URL/Host address: --host-address <ip-address of the 1st Controller Node>

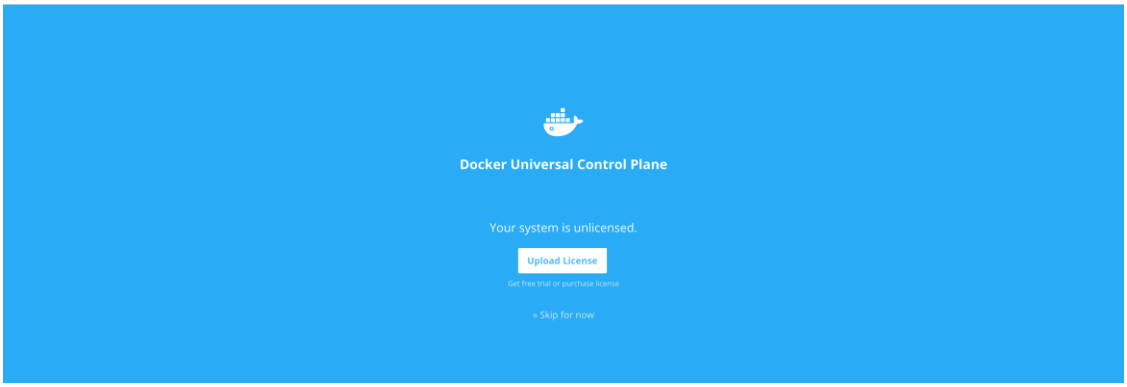
For second architecture, an additional UCP install command parameter '–controller-port' is needed to specify port 4443. This is required as DTR services are co-located on the same nodes and DTR by default uses 443. The command to install UCP controller for second architecture will be:

docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:1.1.4 install --host-address <ip-address of the 1st Controller node --interactive --controller-port 4443

---

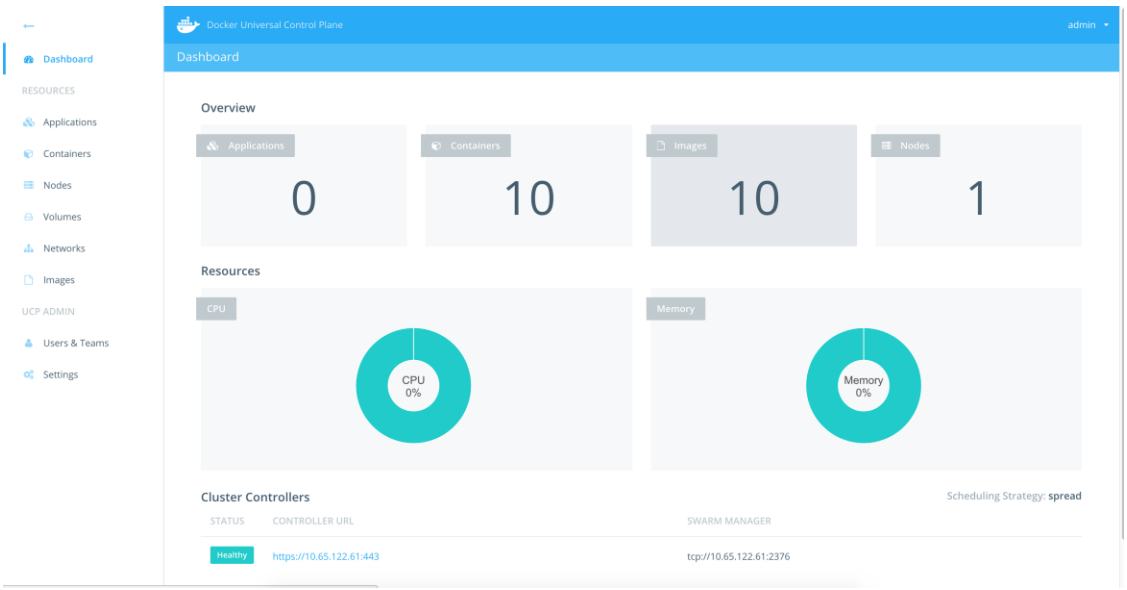
2. Install licenses to the Docker UCP Controller node:





3. Follow the instruction to upload the license file by clicking 'upload License'.

Accessing UCP Dashboard:



This shows that we have one Docker UCP Controller up and running.

For second architecture UCP URL would be – https://<ip-address of the primary controller node>:4443

Backup Controller CAs and Add UCP Replicas

We take a backup of root-CA certificates and copy them on the rest of the nodes. This enables them to join the existing single node cluster securely. After taking backup we can add Controller replicas and UCP nodes.

1. Take backup of root-ca to be used when adding replicas and UCP nodes to form cluster:

```
[root@UCP-Ctrl-1 ~]# docker run --rm -i --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:1.1.4 backup --interactive --root-ca-only --passphrase "se
cret" > /root/backup.tar
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
INFO[0000] We're about to temporarily stop the local UCP CA containers for UCP ID: 4FZN:C6JH:RGST:74ML:DFQ2:EKAE:VQEI:X6KX:IEUD:OZ7M:PBAO:5BWY
Do you want proceed with the backup? (y/n): y
INFO[0001] Temporarily stopping the local CA containers to ensure a consistent backup
INFO[0000] Beginning backup
INFO[0000] Backup completed successfully
INFO[0003] Resuming stopped UCP containers
```

In case the above command fails then we need to pass the cluster-id instead of using --interactive switch in the command. To get the cluster-id run the below commands:

```
[root@UCP-Ctrl-1 ~]# docker run --rm --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:1.1.4 id 1 > /tmp/id
time="2017-01-17T18:01:28Z" level=info msg="Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible"
[root@UCP-Ctrl-1 ~]# cat /tmp/id
BWDO:DSHC:IF7F:J2MV:N56F:5ETJ:LZEM:24GB:GZGI:72SH:ORE2:THK4
```

2. Copy root-ca backup to all other nodes:

```
[root@UCP-Ctrl-1 ~]# ls -ltr
total 804
-rw-r--r--. 1 root root 705956 Sep 14 04:57 kmod-enic-2.3.0.30-rhel7u2.el7.x86_64.rpm
-rw-------. 1 root root   1852 Nov 18 12:46 anaconda-ks.cfg
-rw-------. 1 root root   1945 Nov 18 13:34 initial-setup-ks.cfg
-rw-r--r--. 1 root root  14612 Nov 19 22:51 epel-release-latest-7.noarch.rpm
-rw-r--r--. 1 root root  87323 Nov 24 01:09 backup.tar
[root@UCP-Ctrl-1 ~]# ansible Docker -m copy -a "src=/root/backup.tar dest=/root/backup.tar"
UCP-Ctrl-1 | SUCCESS => {
    "changed": false,
    "checksum": "a8d1266087dbecb3eed3e452c4c13560f68eccd2",
    "dest": "/root/backup.tar",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/root/backup.tar",
    "secontext": "unconfined_u:object_r:admin_home_t:s0",
    "size": 87323,
    "state": "file",
    "uid": 0
}
UCP-Ctrl-3 | SUCCESS => {
    "changed": true,
    "checksum": "a8d1266087dbecb3eed3e452c4c13560f68eccd2",
    "dest": "/root/backup.tar",
    "gid": 0,
    "group": "root",
    "md5sum": "22dba72627eec5562096c521466a4f22",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:admin_home_t:s0",
    "size": 87323,
    "src": "/root/.ansible/tmp/ansible-tmp-1479929957.1-174145191856582/source",
    "state": "file",
    "uid": 0
}
```

3. Login to the UCP Controller node and execute the UCP installation as below:

```
[root@UCP-Ctrl-2 ~]# docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock -v /root/backup.tar:/backup.tar docker/ucp:1.1.4 join --interactive
--replica --passphrase "secret" --fresh-install
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
WARN[0000] Your system uses devicemapper.  We can not accurately detect available storage space.  Please make sure you have at least 3.00 GB available in /var/lib/do
cker
Please enter the URL to your UCP server: https://10.65.122.61:443
UCP server https://10.65.122.61:443
CA Subject: UCP Client Root CA
Serial Number: 50e994a3aaf9b501
SHA-256 Fingerprint=F9:38:76:95:E2:17:80:F7:3F:43:2F:E9:EF:E4:1C:15:13:D6:E5:63:59:64:33:5A:8C:EB:84:D2:E3:4C:F5:31
Do you want to trust this server and proceed with the join? (y/n): y
Please enter your UCP Admin username: admin
Please enter your UCP Admin password:
INFO[0021] All required images are present
We detected the following hostnames/IP addresses for this system [UCP-Ctrl-2.cisco.com 127.0.0.1 172.17.0.1 10.65.122.62]

You may enter additional aliases (SANs) now or press enter to proceed with the above list.
Additional aliases:
INFO[0001] This engine will join UCP and advertise itself with host address 10.65.122.62 - If this is incorrect, please specify an alternative address with the '--ho
st-address' flag
INFO[0001] Verifying your system is compatible with UCP
INFO[0001] Checking that required ports are available and accessible
INFO[0033] Starting local swarm containers
INFO[0036] Starting UCP Controller replica containers
WARN[0089] Configuration updated. You will have to manually restart the docker daemon for the changes to take effect.
WARN[0089] Your cluster only has two controllers. Adding a third node prior to restarting the daemon is strongly recommended to prevent long restart times.
```

This command takes the following parameters −

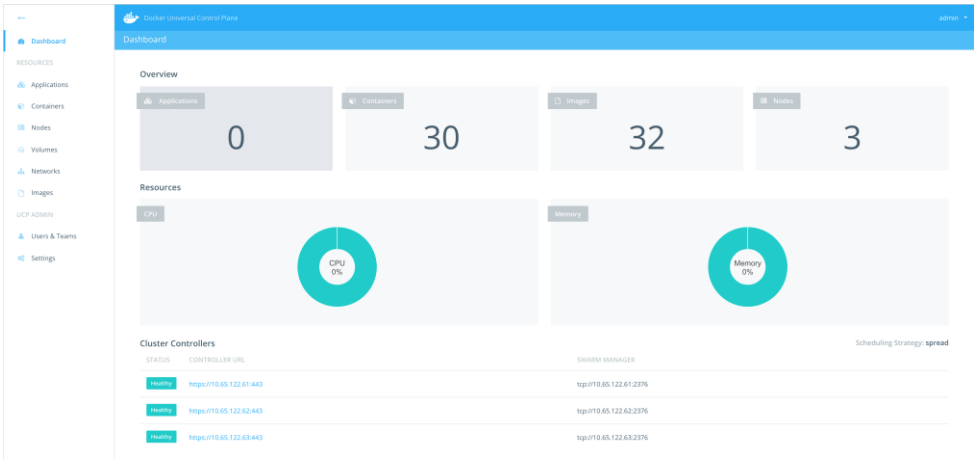
- To join existing cluster − join
- Replica or Primary - --replica
- Root-CA backup - -v /root/backup.tar:/backup.tar
- UCP Server URL − It's the 1st UCP Controller ip address through which we access UCP dashboard − https://<ip-address>:443

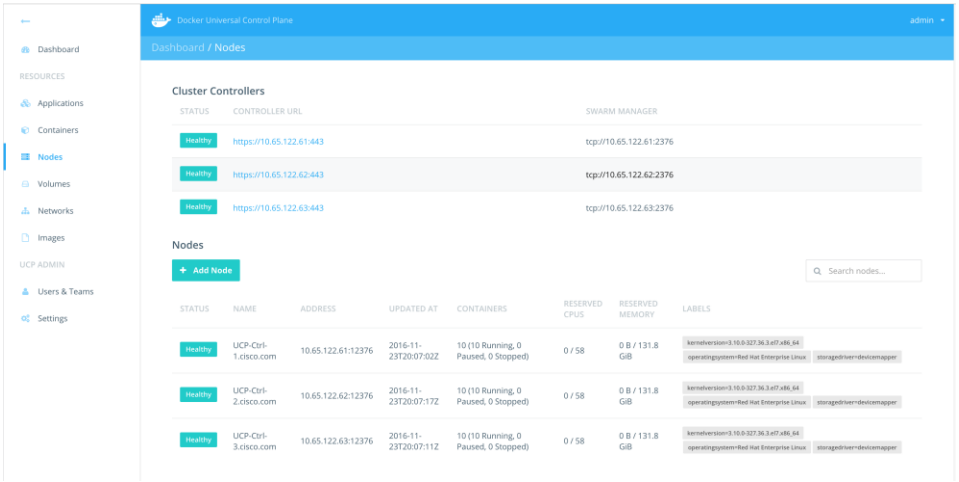
With these switches second controller node will join the first node to form a cluster. We need to follow the same steps for adding the third controller node.

4. To join the third Controller node, we will issue the same command as above:

```
[root@UCP-Ctrl-3 ~]# docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock -v /root/backup.tar:/backup.tar docker/ucp:1.1.4 join --interactive --replica --passph
rase "secret" --fresh-install
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
WARN[0000] Your system uses devicemapper.  We can not accurately detect available storage space.  Please make sure you have at least 3.00 GB available in /var/lib/docker
Please enter the URL to your UCP server: https://10.65.122.61:443
UCP server https://10.65.122.61:443
CA Subject: UCP Client Root CA
Serial Number: 50e994a3aaf9b501
SHA-256 Fingerprint=F9:38:76:95:E2:17:80:F7:3F:43:2F:E9:EF:E4:1C:15:13:D6:E5:63:59:64:33:5A:8C:EB:84:D2:E3:4C:F5:31
Do you want to trust this server and proceed with the join? (y/n): y
Please enter your UCP Admin username: admin
Please enter your UCP Admin password:
INFO[0022] Pulling required images... (this may take a while)
We detected the following hostnames/IP addresses for this system [UCP-Ctrl-3.cisco.com 127.0.0.1 172.17.0.1 10.65.122.63]

You may enter additional aliases (SANs) now or press enter to proceed with the above list.
Additional aliases:
INFO[0001] This engine will join UCP and advertise itself with host address 10.65.122.63 - If this is incorrect, please specify an alternative address with the '--host-address' flag
INFO[0001] Verifying your system is compatible with UCP
INFO[0001] Checking that required ports are available and accessible
INFO[0052] Starting local swarm containers
INFO[0055] Starting UCP Controller replica containers
WARN[0109] Configuration updated. You will have to manually restart the docker daemon for the changes to take effect.
WARN[0109] If you have to restart daemons on multiple controllers, restart them one by one to prevent long restart times.
[root@UCP-Ctrl-3 ~]#
```

5. After the third node has joined we need to restart the Docker services on controller nodes one by one, starting from the first UCP controller node.
6. All the three nodes joins to form a cluster, we will have Docker UCP dashboard status shown as below:



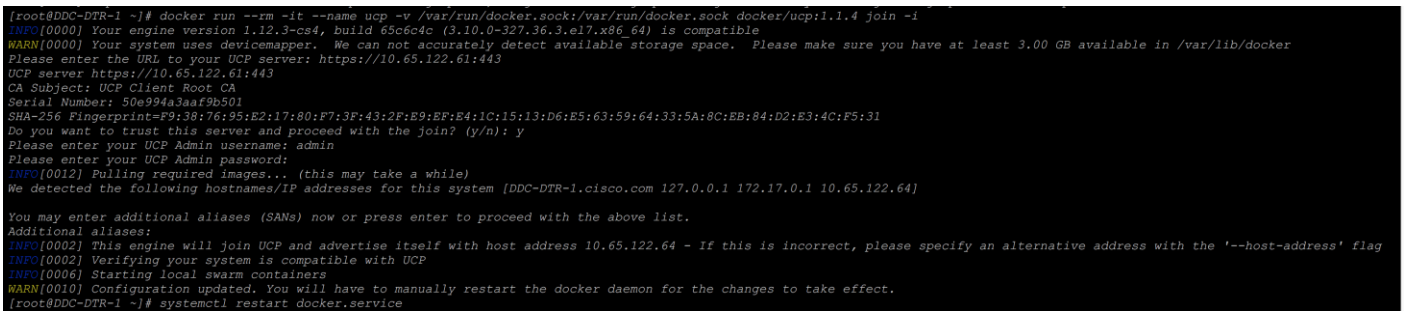


For second architecture UCP Controller replica node join command will be as: docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock -v /root/backup.tar:/backup.tar docker/ucp:1.1.4 join --interactive --replica --passphrase "secret" --fresh-install --controller-port 4443

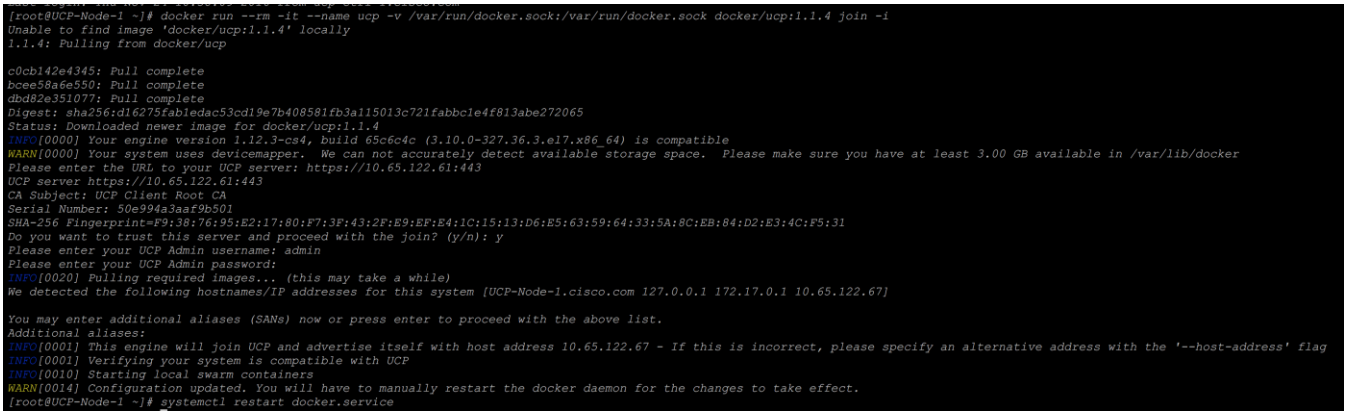
Add UCP Nodes

In this step we will now continue to add rest of the 7 hosts to the newly formed cluster. Even the DTR nodes will be added as UCP nodes first. For all the remaining nodes, command will be same. And here we are not going to give '--replica' to join the nodes to the cluster.

1. Joining the first DTR node to the cluster:



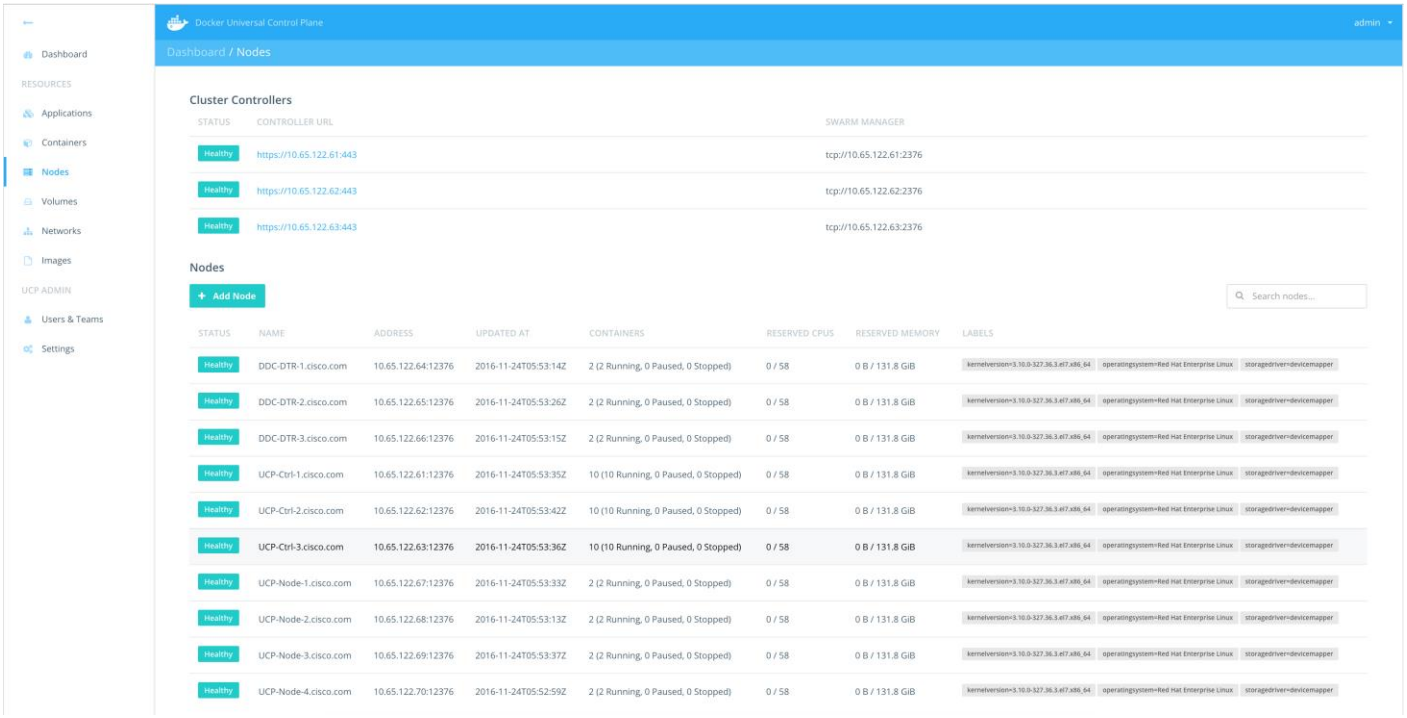
2. Joining the first UCP node to the cluster:

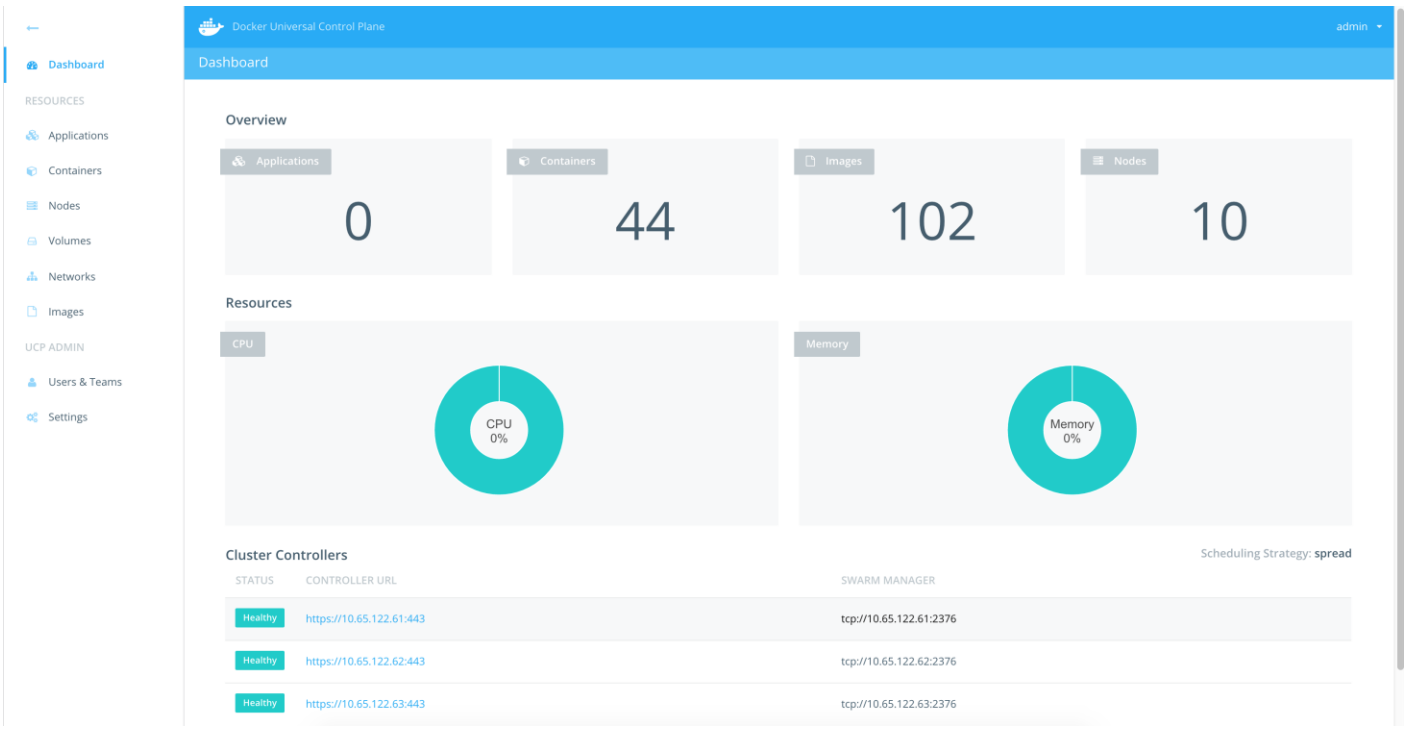


For second architecture UCP server URL will be: https://<ip-address of primary controller node>:4443

3. Similarly we need to add the remaining DTR and UCP nodes by logging into them and running same command as shown above. In all cases UCP server URL remains same. Post installation the Docker services must be started on all the cluster nodes.

After all the nodes join the cluster, the UCP Dash board should look as shown in the screenshot below:





Install and Configure DTR and its Replicas

1. Install DTR application on the designated DTR nodes such as – DTR1, DTR2 and DTR3. Before this we need to open port 80 on all cluster nodes.

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=80/tcp"
DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success
```

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/firewall-cmd --zone=public --add-port=80/tcp --permanent"
UCP-Ctrl-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

UCP-Ctrl-1 | SUCCESS | rc=0 >>
success

UCP-Ctrl-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

UCP-Node-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

UCP-Node-1 | SUCCESS | rc=0 >>
success

UCP-Node-3 | SUCCESS | rc=0 >>
success

UCP-Node-4 | SUCCESS | rc=0 >>
success
```

2. Restart the firewalld services:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/systemctl restart firewalld.service"
DDC-DTR-1 | SUCCESS | rc=0 >>

DDC-DTR-2 | SUCCESS | rc=0 >>

UCP-Ctrl-3 | SUCCESS | rc=0 >>

UCP-Ctrl-2 | SUCCESS | rc=0 >>

UCP-Ctrl-1 | SUCCESS | rc=0 >>

UCP-Node-3 | SUCCESS | rc=0 >>

UCP-Node-2 | SUCCESS | rc=0 >>

UCP-Node-4 | SUCCESS | rc=0 >>

DDC-DTR-3 | SUCCESS | rc=0 >>

UCP-Node-1 | SUCCESS | rc=0 >>
```

3. Download CA certificates on the first UCP Controller node using curl to UCP Controller URL:

```
[root@UCP-Ctrl-1 ucp]# curl -k https://10.65.122.61/ca > ucp-ca.pem
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1842  100  1842    0     0  22434      0 --:--:-- --:--:-- --:--:-- 22740
[root@UCP-Ctrl-1 ucp]# ls -ltr
total 52
-rw-------. 1 root root  1675 Nov 24 05:57 key.pem
-rw-r--r--. 1 root root   572 Nov 24 05:57 env.sh
-rw-r--r--. 1 root root   625 Nov 24 05:57 env.ps1
-rw-r--r--. 1 root root   586 Nov 24 05:57 env.cmd
-rw-r--r--. 1 root root   450 Nov 24 05:57 cert.pub
-rw-r--r--. 1 root root  5392 Nov 24 05:57 cert.pem
-rw-r--r--. 1 root root  3684 Nov 24 05:57 ca.pem
-rw-r--r--. 1 root root 13748 Nov 24 11:38 ucp-bundle-admin.zip
-rw-r--r--. 1 root root  1842 Nov 24 21:47 ucp-ca.pem
```

4. To install DTR application on the DTR nodes, execute the following command on the first UCP Controller Node:

```
[root@UCP-Ctrl-1 ucp]# docker run --rm -it docker/dtr:2.0.3 install --ucp-url https://10.65.122.61 --ucp-node DDC-DTR-1.cisco.com --dtr-external-url 10.65.122.64 --ucp-username admin
--ucp-password Nbv12345! --ucp-ca "$(cat ucp-ca.pem)"
INFO[0000] Beginning Docker Trusted Registry installation
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0000] Checking if the node is okay to install on
INFO[0001] Creating network: DDC-DTR-1.cisco.com/dtr-br
INFO[0002] Connecting to network: DDC-DTR-1.cisco.com/dtr-br
INFO[0002] Waiting for phase2 container to be known to the Docker daemon
INFO[0002] Creating network: dtr-ol
INFO[0003] Connecting to network: dtr-ol
INFO[0003] Waiting for phase2 container to be known to the Docker daemon
INFO[0004] Starting UCP connectivity test
INFO[0004] UCP connectivity test passed
INFO[0004] Setting up replica volumes...
INFO[0005] Creating initial CA certificates
INFO[0052] Waiting for etcd...
INFO[0055] Generated TLS certificate.                    domain=10.65.122.64
INFO[0056] License config copied from UCP.
INFO[0056] Getting container configuration and starting containers...
INFO[0056] Recreating dtr-etcd-ea5f0db3c5fb...
INFO[0064] Recreating dtr-rethinkdb-ea5f0db3c5fb...
INFO[0221] Recreating dtr-registry-ea5f0db3c5fb...
INFO[0395] Recreating dtr-api-ea5f0db3c5fb...
INFO[0594] Recreating dtr-nginx-ea5f0db3c5fb...
INFO[0770] Waiting for etcd...
INFO[0770] Verifying auth settings...
INFO[0770] Waiting for etcd...
INFO[0770] Waiting for DTR to start...
INFO[0775] Waiting for DTR to start...
INFO[0775] Authentication test passed.
INFO[0776] Migrating database schema                     fromVersion=0 toVersion=4
INFO[0780] Migrated database from version 0 to 4
INFO[0780] Installation is complete
INFO[0780] Replica ID is set to: ea5f0db3c5fb
INFO[0780] You can use flag '--existing-replica-id ea5f0db3c5fb' when joining other replicas to your Docker Trusted Registry Cluster
```

DTR installation has to be executed from primary UCP Controller Nodes itself for all DTR nodes. No need to login in to DTR nodes individually and executing install command.

Following command parameters are used in this command:

- Specific DTR versioning tag – docker/dtr:2.0.3
- UCP URL – 1st UCP Controller URL – --ucp-url https://<ip-address>
- UCP Node - --ucp-node <1st DTR Node FQDN>
- DTR External URL - --dtr-external-url <ip address of the 1st DTR Node>
- UCP CA Certs - --ucp-ca <cert file with path where we downloaded certs using curl>

5. Join other DTR Nodes as DTR replicas:

```
[root@UCP-Ctrl-1 ucp]# docker run --rm -it docker/dtr:2.0.3 join --ucp-url https://10.65.122.61 --ucp-node DDC-DTR-2.cisco.com --existing-replica-id ea5f0db3c5fb --ucp-username admin
--ucp-password Nbv12345! --ucp-ca "$(cat ucp-ca.pem)"
INFO[0000] Beginning Docker Trusted Registry replica join
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0004] Validating UCP cert
INFO[0004] UCP cert validation successful
INFO[0004] Checking if the node is okay to install on
INFO[0006] Starting overlay network test
INFO[0020] Overlay networking test passed.
INFO[0020] Creating network: DDC-DTR-2.cisco.com/dtr-br
INFO[0020] Connecting to network: DDC-DTR-2.cisco.com/dtr-br
INFO[0020] Waiting for phase2 container to be known to the Docker daemon
INFO[0022] Connecting to network: dtr-ol
INFO[0022] Waiting for phase2 container to be known to the Docker daemon
INFO[0024] Starting UCP connectivity test
INFO[0024] UCP connectivity test passed
INFO[0024] Setting up replica volumes...
INFO[0024] Connecting to etcd...
INFO[0072] Waiting for etcd...
INFO[0072] Getting container configuration and starting containers...
INFO[0074] Recreating dtr-etcd-cc57c8974d24...
INFO[0081] Recreating dtr-rethinkdb-cc57c8974d24...
INFO[0233] Recreating dtr-registry-cc57c8974d24...
INFO[0403] Recreating dtr-api-cc57c8974d24...
INFO[0602] Recreating dtr-nginx-cc57c8974d24...
INFO[0775] Waiting for etcd...
INFO[0775] Verifying auth settings...
INFO[0775] Waiting for etcd...
INFO[0775] Waiting for DTR to start...
INFO[0776] Authentication test passed.
INFO[0776] Updating replication settings...
WARN[0778] Warning! You are using DTR with the filesystem driver in an HA setup. This will not work with non-clustered file systems. See the documentation for how to configure HA imag
e storage.
INFO[0778] Join is complete
INFO[0778] Replica ID is set to: cc57c8974d24
INFO[0778] There are currently 2 replicas in your Docker Trusted Registry cluster
INFO[0778] You currently have an even number of replicas which can impact cluster availability
INFO[0778] It is recommended that you have 3, 5 or 7 replicas in your cluster
[root@UCP-Ctrl-1 ucp]#
```

Here we have added --existing-replica-id <id as provided at the beginning of the first DTR Node addition>

6. We will now add the DTR application to the third DTR Node:

```
[root@UCP-Ctrl-1 ucp]# docker run --rm -it docker/dtr:2.0.3 join --ucp-url https://10.65.122.61 --ucp-node DDC-DTR-3.cisco.com --existing-replica-id ea5f0db3c5fb --ucp-username admin
--ucp-password Nbv12345! --ucp-ca "$(cat ucp-ca.pem)"
INFO[0000] Beginning Docker Trusted Registry replica join
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0000] Validating UCP cert
INFO[0000] UCP cert validation successful
INFO[0004] Validating UCP cert
INFO[0004] UCP cert validation successful
INFO[0004] Checking if the node is okay to install on
INFO[0005] Starting overlay network test
INFO[0021] Overlay networking test passed.
INFO[0021] Creating network: DDC-DTR-3.cisco.com/dtr-br
INFO[0022] Connecting to network: DDC-DTR-3.cisco.com/dtr-br
INFO[0022] Waiting for phase2 container to be known to the Docker daemon
INFO[0023] Connecting to network: dtr-ol
INFO[0023] Waiting for phase2 container to be known to the Docker daemon
INFO[0026] Starting UCP connectivity test
INFO[0026] UCP connectivity test passed
INFO[0026] Setting up replica volumes...
INFO[0027] Connecting to etcd...
INFO[0072] Waiting for etcd...
INFO[0073] Getting container configuration and starting containers...
INFO[0074] Recreating dtr-etcd-448e20d5d8df...
INFO[0083] Recreating dtr-rethinkdb-448e20d5d8df...
INFO[0236] Recreating dtr-registry-448e20d5d8df...
INFO[0405] Recreating dtr-api-448e20d5d8df...
INFO[0603] Recreating dtr-nginx-448e20d5d8df...
INFO[0779] Waiting for etcd...
INFO[0780] Verifying auth settings...
INFO[0780] Waiting for etcd...
INFO[0780] Waiting for DTR to start...
INFO[0780] Authentication test passed.
INFO[0780] Updating replication settings...
WARN[0783] Warning! You are using DTR with the filesystem driver in an HA setup. This will not work with non-clustered file systems. See the documentation for how to configure HA imag
e storage.
INFO[0783] Join is complete
INFO[0783] Replica ID is set to: 448e20d5d8df
INFO[0783] There are currently 3 replicas in your Docker Trusted Registry cluster
```

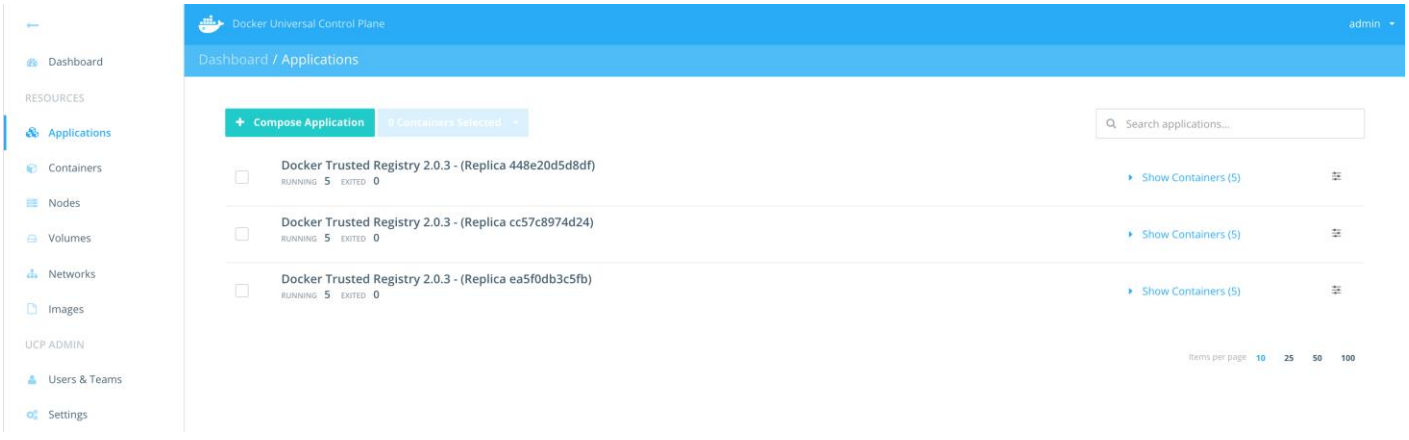
For second architecture DTR and its replica installation commands for 1st node: docker run --rm -it docker/dtr:2.0.3 install --ucp-url https://<UCP Primary Controller ip-address>:4443 --ucp-node UCP-Ctrl-1.cisco.com --dtr-external-url <1st DTR node ip-address> --ucp-username <admin-username> --ucp-password <UCP admin password> --ucp-ca "$(cat ucp-ca.pem)"

DTR replica on 2nd node: docker run --rm -it docker/dtr:2.0.3 join --ucp-url https://<UCP Primary Controller ip-address>:4443 --ucp-node UCP-Ctrl-2.cisco.com --existing-replica-id <replica id as given out in above command output> --ucp-username <admin-username> --ucp-password <UCP admin password> --ucp-ca "$(cat ucp-ca.pem)"

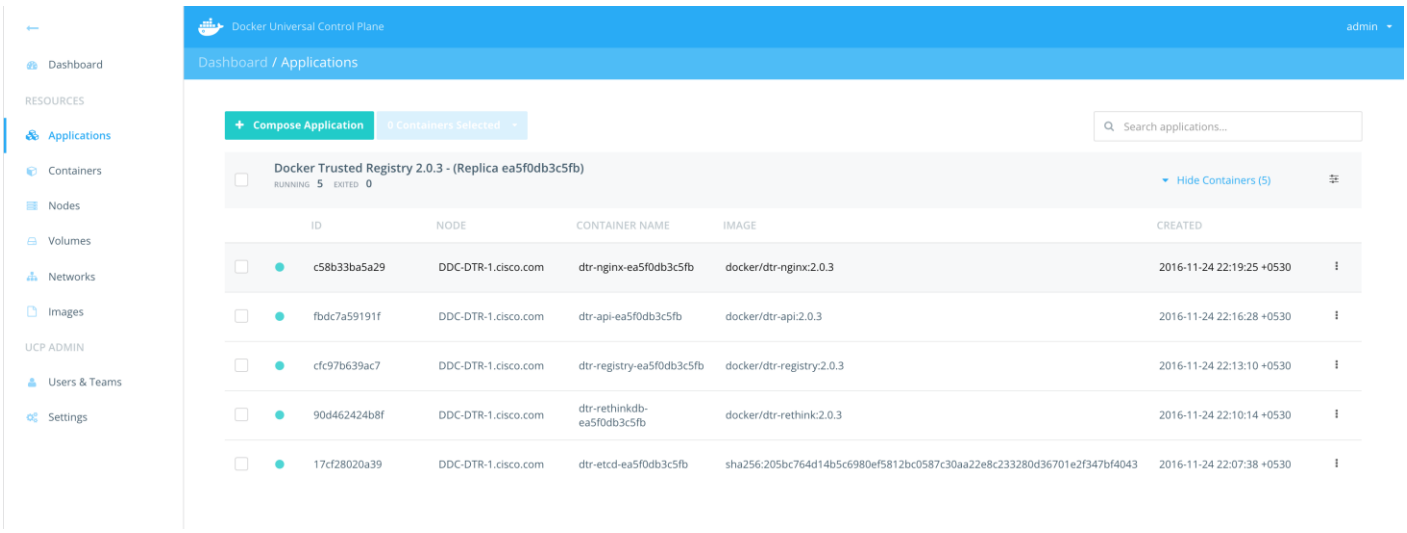
DTR replica on 3rd node : docker run --rm -it docker/dtr:2.0.3 join --ucp-url https://<UCP Primary Controller ip-address>:4443 --ucp-node UCP-Ctrl-3.cisco.com --existing-replica-id <replica id as above> --ucp-username <admin-username> --ucp-password <UCP admin password> --ucp-ca "$(cat ucp-ca.pem)"

---

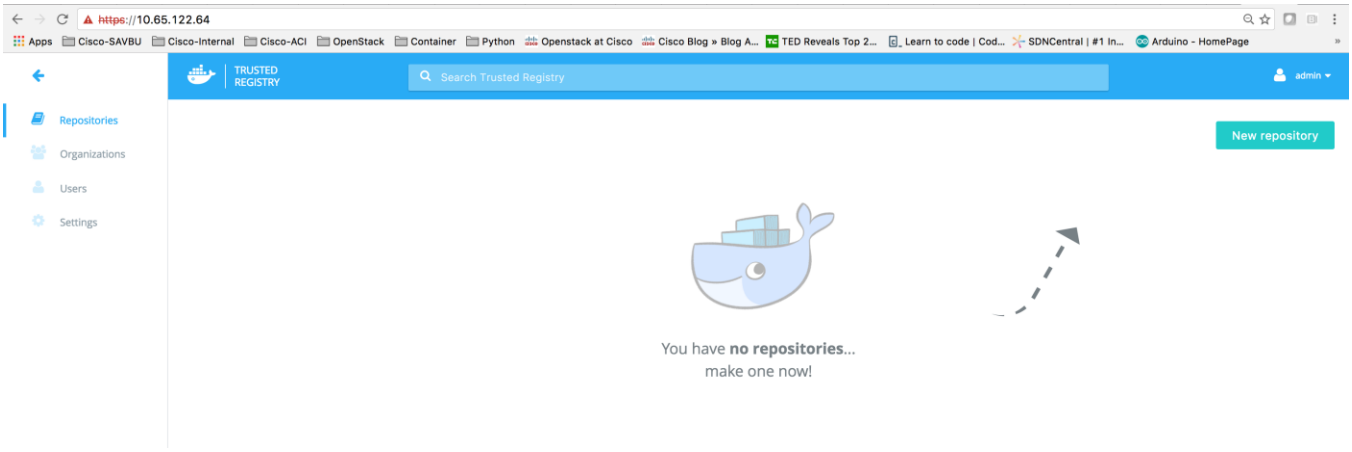
7. UCP dashboard will show three DTR applications getting installed on three DTR nodes as below:



8. The DTR containers running on DTR Nodes are:



9. Login to DTR URL (which is the first DTR Node ip address):



NFS shared volume configuration for the DTR nodes are outlined below. NFS backend and configuration details have been omitted for simplicity.

10. Next step is to configure NFS file system on all three DTR Nodes to have the shared access.
11. Open firewall ports for NFS services on DTR nodes only:

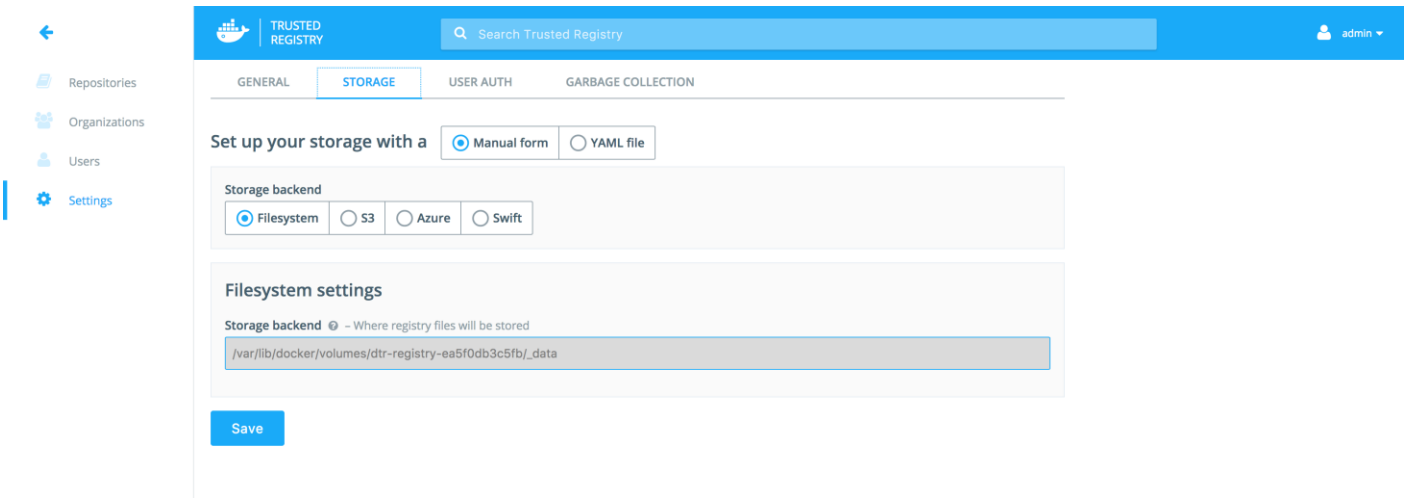
```
[root@UCP-Ctrl-1 ~]# ansible DTR -a "/usr/bin/firewall-cmd --zone=public --add-service=nfs"
DDC-DTR-1 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

[root@UCP-Ctrl-1 ~]# ansible DTR -a "/usr/bin/firewall-cmd --zone=public --add-service=mountd"
DDC-DTR-3 | SUCCESS | rc=0 >>
success

DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success

[root@UCP-Ctrl-1 ~]# ansible DTR -a "/usr/bin/firewall-cmd --zone=public --add-service=rpc-bind"
DDC-DTR-2 | SUCCESS | rc=0 >>
success

DDC-DTR-3 | SUCCESS | rc=0 >>
success

DDC-DTR-1 | SUCCESS | rc=0 >>
success
```

12. Mount external NFS file share on all the three DTR Nodes as shown below:

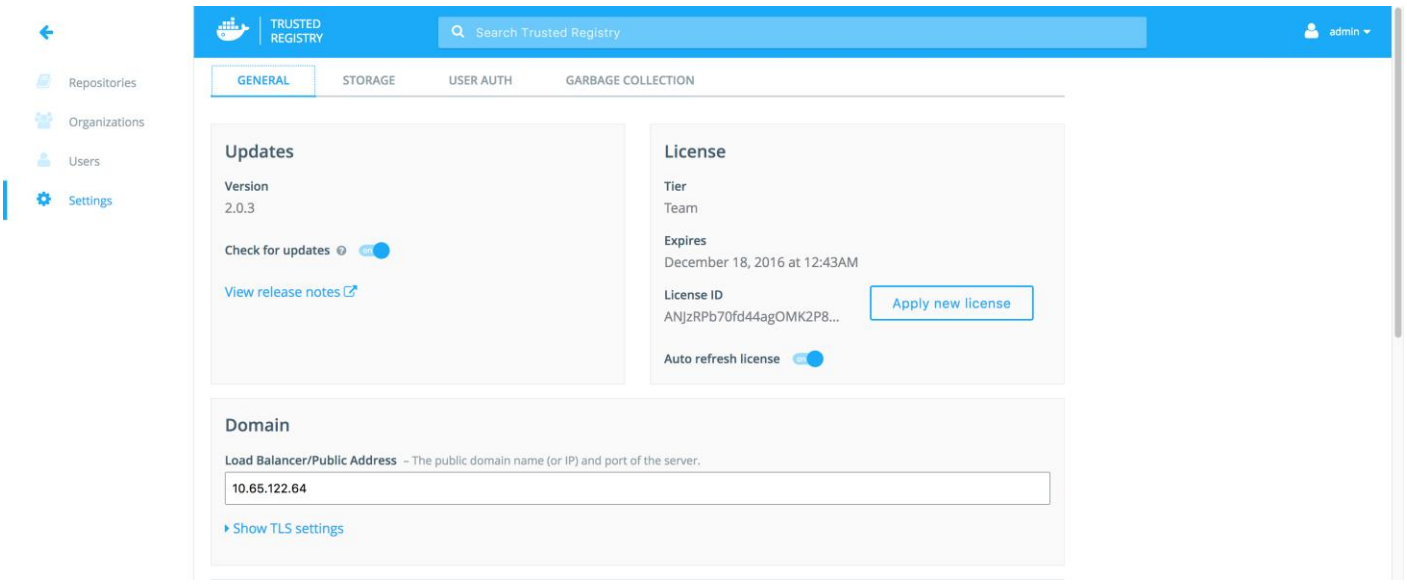
```
[root@DDC-DTR-1 ~]# #mount -t nfs 10.65.122.80:/DTR-NFS /var/lib/docker/volumes/dtr-registry-ea5f0db3c5fb/_data/
[root@DDC-DTR-1 ~]# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Sat Nov 19 12:22:16 2016
#
# 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=a4947bd7-0e24-4b39-8694-cc52945e85a2 /boot                   xfs     defaults        0 0
/dev/mapper/rhel-home   /home                   xfs     defaults        0 0
/dev/mapper/rhel-swap   swap                    swap    defaults        0 0
10.65.122.80:/DTR-NFS  /var/lib/docker/volumes/dtr-registry-ea5f0db3c5fb/_data  nfs     defaults 0 0
[root@DDC-DTR-1 ~]#
```

NFS mount point should be /var/lib/docker/volumes/dtr-registry-<DTR application id>/_data

13. Once the NFS mount configured, DTR dashboard reflects it automatically:

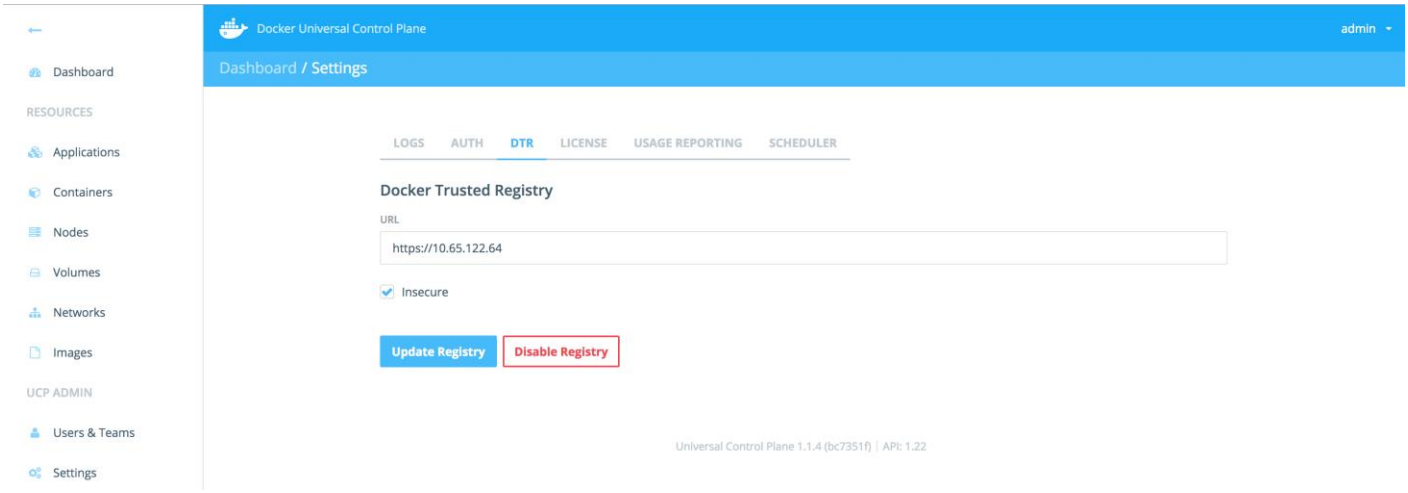


14. Configure the domain in the DTR dashboard:



Docker recommends using external load balancer VIP address here. As external load balancer is out of scope in this solution, we have mentioned the first DTR node's ip address.

15. Configure DTR domain in the UCP Cluster:



Validate Docker UCP and DTR Cluster

This section validates the Docker UCP/DTR cluster deployment. To check the basic sanity follow these steps:

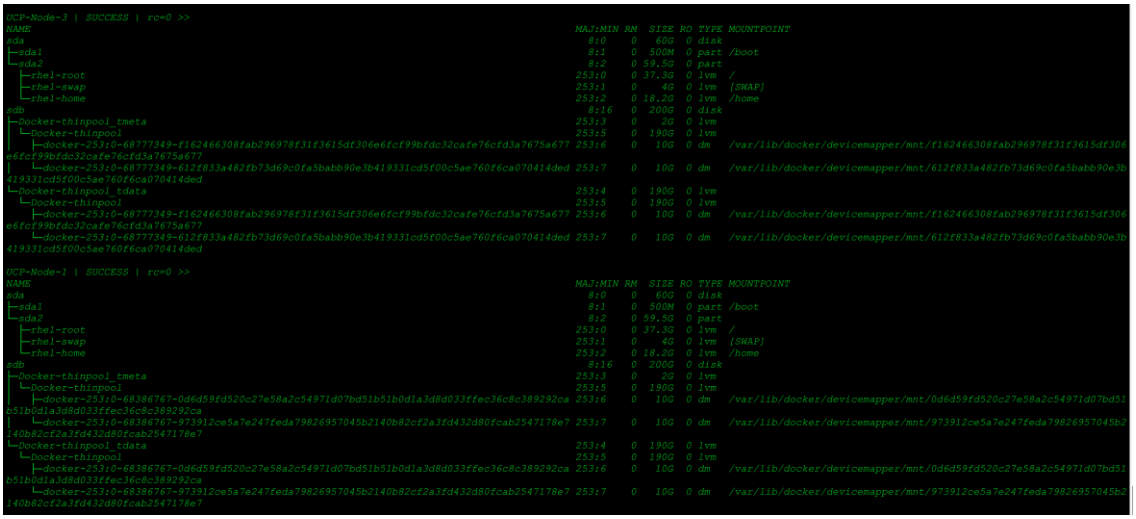
1. Run 'docker ps' to see the status of all infrastructure containers and DTR application containers. There should not be any stopped and/or exited containers:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/docker ps"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS
              NAMES
aa671564145b        docker/ucp-controller:1.1.4    "/bin/controller serv"   36 hours ago        Up 26 hours         0.0.0.0:443->8080/tcp
              ucp-controller
a43a2d26e95a        docker/ucp-auth:1.1.4          "/usr/local/bin/enzi "   36 hours ago        Up 26 hours         0.0.0.0:12386->4443/tcp
              ucp-auth-worker
d1502ab5ebef        docker/ucp-auth:1.1.4          "/usr/local/bin/enzi "   36 hours ago        Up 26 hours         0.0.0.0:12385->4443/tcp
              ucp-auth-api
f6cd735bc08f        docker/ucp-auth-store:1.1.4    "rethinkdb --bind all"   36 hours ago        Up 26 hours         0.0.0.0:12383-12384->12383-12384/tcp
              ucp-auth-store
b70accdb9c27        docker/ucp-cfssl:1.1.4         "/bin/cfssl serve -ad"   36 hours ago        Up 26 hours         8888/tcp, 0.0.0.0:12381->12381/tcp
              ucp-cluster-root-ca
782dd3807408        docker/ucp-cfssl:1.1.4         "/bin/cfssl serve -ad"   36 hours ago        Up 26 hours         8888/tcp, 0.0.0.0:12382->12382/tcp
              ucp-client-root-ca
afaf7b835119        docker/ucp-swarm:1.1.4         "/swarm manage --tlsv"   36 hours ago        Up 26 hours         0.0.0.0:2376->2375/tcp
              ucp-swarm-manager
37cc6d98b7f6        docker/ucp-swarm:1.1.4         "/swarm join --discov"   36 hours ago        Up 26 hours         2375/tcp
              ucp-swarm-join
b947c1d6d505        docker/ucp-proxy:1.1.4         "/bin/run"               36 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp
              ucp-proxy
196563ebe0bb        docker/ucp-etcd:1.1.4          "/bin/etcd --data-dir"   36 hours ago        Up 26 hours         2380/tcp, 4001/tcp, 7001/tcp, 0.0.0.0:12380->12380/tcp, 0.0.0.0:1237
9->2379/tcp   ucp-kv

DDC-DTR-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                        COMMAND                  CREATED             STATUS              PORTS                                        NAMES
94ad6d6860c2        docker/dtr-nginx:2.0.3       "/init --skip-runit /"   14 hours ago        Up 14 hours         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp     dtr-nginx-cc57c8974d24
1daa74c82d28        docker/dtr-api:2.0.3         "/bin/server"            14 hours ago        Up 14 hours         80/tcp                                       dtr-api-cc57c8974d24
9c49956dfb18        docker/dtr-registry:2.0.3    "/init --skip-runit /"   14 hours ago        Up 14 hours         3000/tcp                                     dtr-registry-cc57c8974d24
7faaf86ec68f        docker/dtr-rethink:2.0.3     "/start.sh"              14 hours ago        Up 14 hours         8080/tcp                                     dtr-rethinkdb-cc57c8974d24
3cb0fd285daa        205bc764d14b                 "/etcd"                  14 hours ago        Up 14 hours         2379-2380/tcp, 4001/tcp, 7001/tcp            dtr-etcd-cc57c8974d24
e74563af7f91        docker/ucp-swarm:1.1.4       "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                                     ucp-swarm-join
1d0856dcfaf7        docker/ucp-proxy:1.1.4       "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp                      ucp-proxy

DDC-DTR-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                        COMMAND                  CREATED             STATUS              PORTS                                        NAMES
c58b33ba5a29        docker/dtr-nginx:2.0.3       "/init --skip-runit /"   15 hours ago        Up 15 hours         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp     dtr-nginx-ea5f0db3c5fb
fbdc7a59191f        docker/dtr-api:2.0.3         "/bin/server"            15 hours ago        Up 15 hours         80/tcp                                       dtr-api-ea5f0db3c5fb
cfc97b639ac7        docker/dtr-registry:2.0.3    "/init --skip-runit /"   15 hours ago        Up 15 hours         3000/tcp                                     dtr-registry-ea5f0db3c5fb
90d462424b8f        docker/dtr-rethink:2.0.3     "/start.sh"              15 hours ago        Up 15 hours         8080/tcp                                     dtr-rethinkdb-ea5f0db3c5fb
17cf28020a39        205bc764d14b                 "/etcd"                  15 hours ago        Up 15 hours         2379-2380/tcp, 4001/tcp, 7001/tcp            dtr-etcd-ea5f0db3c5fb
1c20e20be40f        docker/ucp-swarm:1.1.4       "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                                     ucp-swarm-join
9f0bae8a0d67        docker/ucp-proxy:1.1.4       "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp                      ucp-proxy
```

```
UCP-Node-4 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                     NAMES
a2b3b7e5514f        docker/ucp-swarm:1.1.4   "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                  ucp-swarm-join
11c3222be8bf        docker/ucp-proxy:1.1.4   "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp   ucp-proxy

UCP-Node-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                     NAMES
4984eeabf238        docker/ucp-swarm:1.1.4   "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                  ucp-swarm-join
43a27493cb92        docker/ucp-proxy:1.1.4   "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp   ucp-proxy

UCP-Node-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                     NAMES
68b8793ce514        docker/ucp-swarm:1.1.4   "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                  ucp-swarm-join
81ed94d768a0        docker/ucp-proxy:1.1.4   "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp   ucp-proxy

UCP-Node-3 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                     NAMES
a4fe4f989e53        docker/ucp-swarm:1.1.4   "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                  ucp-swarm-join
eb2c68059228        docker/ucp-proxy:1.1.4   "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp   ucp-proxy
```

2. Verify the Docker device-mapper in direct-lvm mode is functioning correctly and containers are getting thin storage provisioned via Docker thinpool:

```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/lsblk"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
NAME                                                                                              MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                                                                                                 8:0    0   60G  0 disk
├─sda1                                                                                              8:1    0  500M  0 part /boot
└─sda2                                                                                              8:2    0 59.5G  0 part
  ├─rhel-root                                                                                     253:0    0 37.3G  0 lvm  /
  ├─rhel-swap                                                                                     253:1    0    4G  0 lvm  [SWAP]
  └─rhel-home                                                                                     253:4    0 18.2G  0 lvm  /home
sdb                                                                                                 8:16   0  200G  0 disk
├─Docker-thinpool_tmeta                                                                           253:2    0    2G  0 lvm
│ └─Docker-thinpool                                                                               253:5    0  190G  0 lvm
│   ├─docker-253:0-1459285-d078356a501a7c310b68ac479801122d49ce470743d346a553ce110446dc022b 253:6    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/d078356a501a7c310b68ac479801122d4
9ce470743d346a553ce110446dc022b
│   ├─docker-253:0-1459285-9c6c727565e4dd8cc4717dc104e34b4abf1ae4070d1a417e9a92a5347da1a042 253:7    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/9c6c727565e4dd8cc4717dc104e34b4ab
f1ae4070d1a417e9a92a5347da1a042
│   ├─docker-253:0-1459285-7e6ba8eecc6e6dd872e95d6a9a45864787cef1a373a16fa75410ea22d5ca1cf0 253:8    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/7e6ba8eecc6e6dd872e95d6a9a4586478
7cef1a373a16fa75410ea22d5ca1cf0
│   ├─docker-253:0-1459285-1d963c4153462fb1975285ca6e6904e4d23d3e8d85605c7ab9a34c5e708f98b1 253:9    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/1d963c4153462fb1975285ca6e6904e4d
23d3e8d85605c7ab9a34c5e708f98b1
│   ├─docker-253:0-1459285-2a0b5ebf36c6a27e59b6e7876ddf291917f0e63ef4e5b014931b41452fb3706a 253:10   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/2a0b5ebf36c6a27e59b6e7876ddf29191
7f0e63ef4e5b014931b41452fb3706a
│   ├─docker-253:0-1459285-0edb3c4b2bc130c1da493019e93207ce86f13b0f1ffebdb4dcdf84ab8568ee07 253:11   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/0edb3c4b2bc130c1da493019e93207ce8
6f13b0f1ffebdb4dcdf84ab8568ee07
│   ├─docker-253:0-1459285-75b6d7786e04c037bdb0dfd59ca30367de842b1aa7976d5f12b8b7e62f82b324 253:12   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/75b6d7786e04c037bdb0dfd59ca30367d
e842b1aa7976d5f12b8b7e62f82b324
│   ├─docker-253:0-1459285-4ff9c3d145d27d58baf29ba5d154be94c55ac4ee06ea15e60a2df2384fad47e2 253:13   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/4ff9c3d145d27d58baf29ba5d154be94c
55ac4ee06ea15e60a2df2384fad47e2
│   ├─docker-253:0-1459285-3487171c77322aaa1592b8d276fb82e81278aa3887825bb62b8c19ef23bf0f36 253:14   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/3487171c77322aaa1592b8d276fb82e81
278aa3887825bb62b8c19ef23bf0f36
│   └─docker-253:0-1459285-50010a1e977e252692b7a1dfa958ab1d6963b28370ee17d67e1acf70183fa366 253:15   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/50010a1e977e252692b7a1dfa958ab1d6
963b28370ee17d67e1acf70183fa366
└─Docker-thinpool_tdata                                                                           253:3    0  190G  0 lvm
  └─Docker-thinpool                                                                               253:5    0  190G  0 lvm
    ├─docker-253:0-1459285-d078356a501a7c310b68ac479801122d49ce470743d346a553ce110446dc022b 253:6    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/d078356a501a7c310b68ac479801122d4
9ce470743d346a553ce110446dc022b
    ├─docker-253:0-1459285-9c6c727565e4dd8cc4717dc104e34b4abf1ae4070d1a417e9a92a5347da1a042 253:7    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/9c6c727565e4dd8cc4717dc104e34b4ab
f1ae4070d1a417e9a92a5347da1a042
    ├─docker-253:0-1459285-7e6ba8eecc6e6dd872e95d6a9a45864787cef1a373a16fa75410ea22d5ca1cf0 253:8    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/7e6ba8eecc6e6dd872e95d6a9a4586478
7cef1a373a16fa75410ea22d5ca1cf0
    ├─docker-253:0-1459285-1d963c4153462fb1975285ca6e6904e4d23d3e8d85605c7ab9a34c5e708f98b1 253:9    0   10G  0 dm   /var/lib/docker/devicemapper/mnt/1d963c4153462fb1975285ca6e6904e4d
23d3e8d85605c7ab9a34c5e708f98b1
    ├─docker-253:0-1459285-2a0b5ebf36c6a27e59b6e7876ddf291917f0e63ef4e5b014931b41452fb3706a 253:10   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/2a0b5ebf36c6a27e59b6e7876ddf29191
7f0e63ef4e5b014931b41452fb3706a
    ├─docker-253:0-1459285-0edb3c4b2bc130c1da493019e93207ce86f13b0f1ffebdb4dcdf84ab8568ee07 253:11   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/0edb3c4b2bc130c1da493019e93207ce8
6f13b0f1ffebdb4dcdf84ab8568ee07
    ├─docker-253:0-1459285-75b6d7786e04c037bdb0dfd59ca30367de842b1aa7976d5f12b8b7e62f82b324 253:12   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/75b6d7786e04c037bdb0dfd59ca30367d
e842b1aa7976d5f12b8b7e62f82b324
    ├─docker-253:0-1459285-4ff9c3d145d27d58baf29ba5d154be94c55ac4ee06ea15e60a2df2384fad47e2 253:13   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/4ff9c3d145d27d58baf29ba5d154be94c
55ac4ee06ea15e60a2df2384fad47e2
    ├─docker-253:0-1459285-3487171c77322aaa1592b8d276fb82e81278aa3887825bb62b8c19ef23bf0f36 253:14   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/3487171c77322aaa1592b8d276fb82e81
278aa3887825bb62b8c19ef23bf0f36
    └─docker-253:0-1459285-50010a1e977e252692b7a1dfa958ab1d6963b28370ee17d67e1acf70183fa366 253:15   0   10G  0 dm   /var/lib/docker/devicemapper/mnt/50010a1e977e252692b7a1dfa958ab1d6
963b28370ee17d67e1acf70183fa366
```



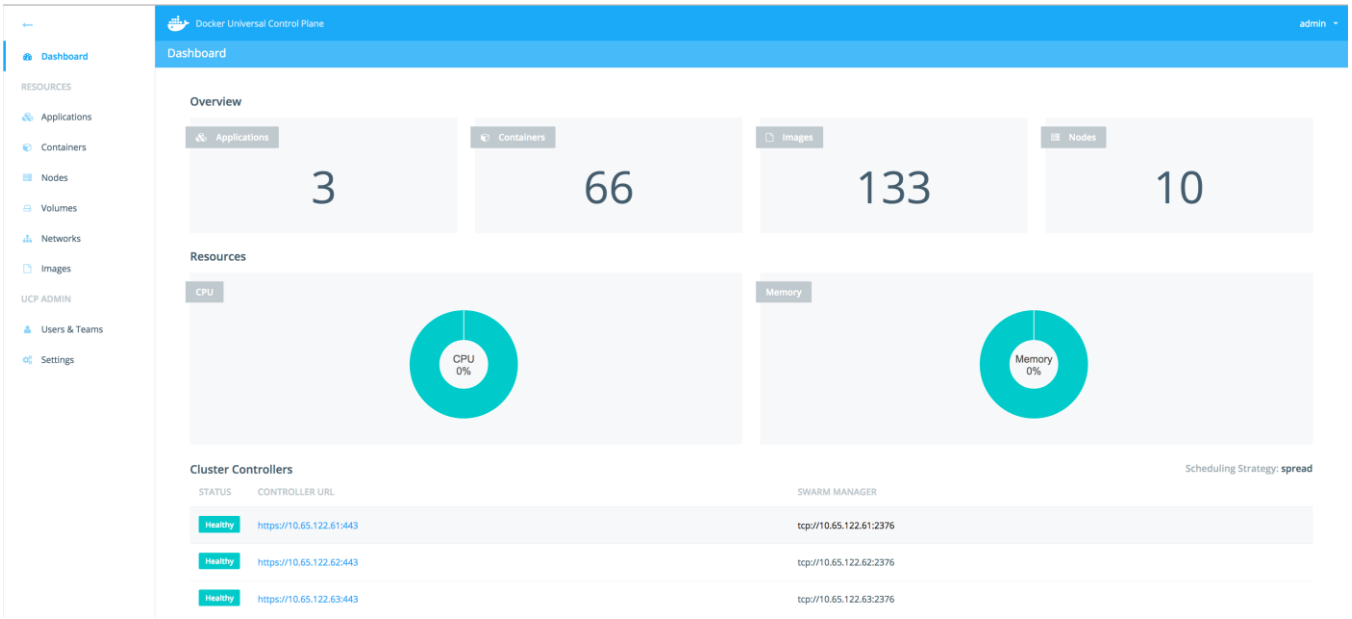
Management and Monitoring Interface

Docker Universal Control Plane (UCP) is a Docker Datacenter component, which provides a GUI dashboard and CL interface to manage and monitor the complete cluster nodes in a single plane of management. Docker UCP is installed as an infrastructure service container on all the designated UCP Controller nodes and it is responsible for managing the entire cluster. UCP can be made highly available through configuring an external load balancer. External load balancer keeps the dashboard available in the event of failure of any of the UCP Controller nodes by assigning a VIP that front ends the individual UCP URLs running on UCP Controller nodes. UCP also integrates DTR services and presents them as an application container in the Docker UCP GUI dashboard. The dashboard covers entire cluster lifecycle management as well as containers, images, network and volumes. Management interface enables you to fetch container images to the cluster from where the containers can be deployed using network/storage plugin of user choice. Also, this provides an option to start/stop, kill and remove containers and their associated images, network and volumes. UCP also does application container lifecycle management along with cluster management.

Universal Control Plane has a built-in security and access control. It supports integration options with LDAP and Active Directory. Also, supports Role Based Access Control (RBAC). Only authorized users can access and make changes to the cluster.

Health status and associated monitoring data gets refreshed periodically and gets updated on the dash board accordingly.

GUI Example



CLI Example

```
# docker info
Containers: 66
 Running: 63
 Paused: 0
 Stopped: 3
Images: 133
Server Version: swarm/1.2.5
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint
Nodes: 10
 DDC-DTR-1.cisco.com: 10.65.122.64:12376
  └ ID: 7TYN:HVC5:XWFI:SH7Z:AYYW:2GZX:MFLK:TG72:QF4E:BLOH:ZOBX:76EJ
  └ Status: Healthy
  └ Containers: 7 (7 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper
  └ UpdatedAt: 2017-02-02T08:36:32Z
```

```
  └ ServerVersion: 1.12.3-cs4

 DDC-DTR-2.cisco.com: 10.65.122.65:12376

  └ ID: UFDJ:UFNQ:47IE:MIOF:RTNF:UX66:CKJ4:6M2O:5EUC:CRCN:UA4U:CFYH

  └ Status: Healthy

  └ Containers: 7 (7 Running, 0 Paused, 0 Stopped)

  └ Reserved CPUs: 0 / 58

  └ Reserved Memory: 0 B / 131.8 GiB

  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper

  └ UpdatedAt: 2017-02-02T08:36:59Z

  └ ServerVersion: 1.12.3-cs4

 DDC-DTR-3.cisco.com: 10.65.122.66:12376

  └ ID: IUPK:LN4O:PD2N:2TXN:QTDC:ZE6L:YQVB:XJV6:WW6W:GI4M:EYO6:KIBE

  └ Status: Healthy

  └ Containers: 7 (7 Running, 0 Paused, 0 Stopped)

  └ Reserved CPUs: 0 / 58

  └ Reserved Memory: 0 B / 131.8 GiB

  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper

  └ UpdatedAt: 2017-02-02T08:36:46Z

  └ ServerVersion: 1.12.3-cs4

 UCP-Ctrl-1.cisco.com: 10.65.122.61:12376

  └ ID: 3OPE:S2YE:KLPG:GMDH:XTW6:IBHD:RYFF:2XSE:WKZP:BDTP:KDW4:T7NA

  └ Status: Healthy

  └ Containers: 10 (10 Running, 0 Paused, 0 Stopped)

  └ Reserved CPUs: 0 / 58

  └ Reserved Memory: 0 B / 131.8 GiB

  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper

  └ UpdatedAt: 2017-02-02T08:36:45Z

  └ ServerVersion: 1.12.3-cs4

 UCP-Ctrl-2.cisco.com: 10.65.122.62:12376

  └ ID: 7KIA:NZBQ:NMYT:LMYL:CZBD:TWE2:JTHK:45HX:YBWW:KOHC:TEGO:3GQO

  └ Status: Healthy

  └ Containers: 11 (11 Running, 0 Paused, 0 Stopped)
```

└ Reserved CPUs: 0 / 58

└ Reserved Memory: 0 B / 131.8 GiB

└ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper

└ UpdatedAt: 2017-02-02T08:36:37Z

└ ServerVersion: 1.12.3-cs4

UCP-Ctrl-3.cisco.com: 10.65.122.63:12376

└ ID: KU2O:W4OU:TGHX:DGU7:7A2I:J44S:J7ZT:D7LD:HSBI:HCS3:BFEO:HFMV

└ Status: Healthy

└ Containers: 10 (10 Running, 0 Paused, 0 Stopped)

└ Reserved CPUs: 0 / 58

└ Reserved Memory: 0 B / 131.8 GiB

└ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper

└ UpdatedAt: 2017-02-02T08:36:24Z

└ ServerVersion: 1.12.3-cs4

UCP-Node-1.cisco.com: 10.65.122.67:12376

└ ID: D4F5:NHRE:SA3H:7IZ3:DTYK:HG6V:MJFQ:B446:GX5G:QRLS:6MVG:LYVM

└ Status: Healthy

└ Containers: 4 (3 Running, 0 Paused, 1 Stopped)

└ Reserved CPUs: 0 / 58

└ Reserved Memory: 0 B / 131.8 GiB

└ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper

└ UpdatedAt: 2017-02-02T08:36:41Z

└ ServerVersion: 1.12.3-cs4

UCP-Node-2.cisco.com: 10.65.122.68:12376

└ ID: 7SHV:S6YR:KTCX:Q722:YBHA:WSBV:QRGE:UR3C:34AC:JTUU:Z2PN:LFGK

└ Status: Healthy

└ Containers: 3 (2 Running, 0 Paused, 1 Stopped)

└ Reserved CPUs: 0 / 58

└ Reserved Memory: 0 B / 131.8 GiB

└ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper

└ UpdatedAt: 2017-02-02T08:36:59Z

└ ServerVersion: 1.12.3-cs4

```
 UCP-Node-3.cisco.com: 10.65.122.69:12376
  └ ID: OG2S:KTYE:6TYS:LUKK:5NAC:OWDI:6FM2:2ENB:ESAS:7BS5:NDYY:S2LA
  └ Status: Healthy
  └ Containers: 3 (3 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper
  └ UpdatedAt: 2017-02-02T08:36:08Z
  └ ServerVersion: 1.12.3-cs4
 UCP-Node-4.cisco.com: 10.65.122.70:12376
  └ ID: 7VTY:5U5D:24QE:VCRY:OOA3:AD2D:RE3H:6YSA:RXR4:ELXD:E46H:YDSG
  └ Status: Healthy
  └ Containers: 4 (3 Running, 0 Paused, 1 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise
Linux, storagedriver=devicemapper
  └ UpdatedAt: 2017-02-02T08:36:43Z
  └ ServerVersion: 1.12.3-cs4
Cluster Managers: 3
 10.65.122.61: Healthy
  └ Orca Controller: https://10.65.122.61:443
  └ Swarm Manager: tcp://10.65.122.61:2376
  └ KV: etcd://10.65.122.61:12379
 10.65.122.62: Healthy
  └ Orca Controller: https://10.65.122.62:443
  └ Swarm Manager: tcp://10.65.122.62:2376
  └ KV: etcd://10.65.122.62:12379
 10.65.122.63: Healthy
  └ Orca Controller: https://10.65.122.63:443
  └ Swarm Manager: tcp://10.65.122.63:2376
  └ KV: etcd://10.65.122.63:12379
Plugins:
 Volume:
```

```
 Network:
Swarm:
 NodeID:
 Is Manager: false
 Node Address:
Security Options:
Kernel Version: 3.10.0-327.36.3.el7.x86_64
Operating System: linux
Architecture: amd64
CPUs: 580
Total Memory: 1.287 TiB
Name: ucp-controller-UCP-Ctrl-1.cisco.com
ID: EATX:GRQR:KCHI:I7FG:SI5Q:YQ6Q:G6RE:6LBB:2XIX:JY6R:D3BU:4KLI
Docker Root Dir:
Debug Mode (client): false
Debug Mode (server): false
Labels:
 com.docker.ucp.license_key=wasvIv_S3w_xAfArYYf31vO6TIXfzD31M0gStiRLSOS5
 com.docker.ucp.license_max_engines=20
 com.docker.ucp.license_expires=2018-01-06 23:47:53 +0000 UTC
```

Solution Validation

Application Container Deployment

To validate this solution we have performed some feature functional tests, high-availability tests, stack scale-up and scale-down tests. Feature functional tests included routine container life-cycle management operations through Docker UCP client bundle and GUI. Basic operation covers, creating containers, start/stop of containers, killing containers, creating network, and using the created networks for running containers. Docker UCP client requires Docker Toolbox to be installed on older versions of Mac or Windows client machine. Docker Toolbox provides tools such as Docker Compose, Docker command line environment along with the others. The following test validation tasks were accomplished using these tools whenever required. Docker Toolbox can be obtained from below URLs:

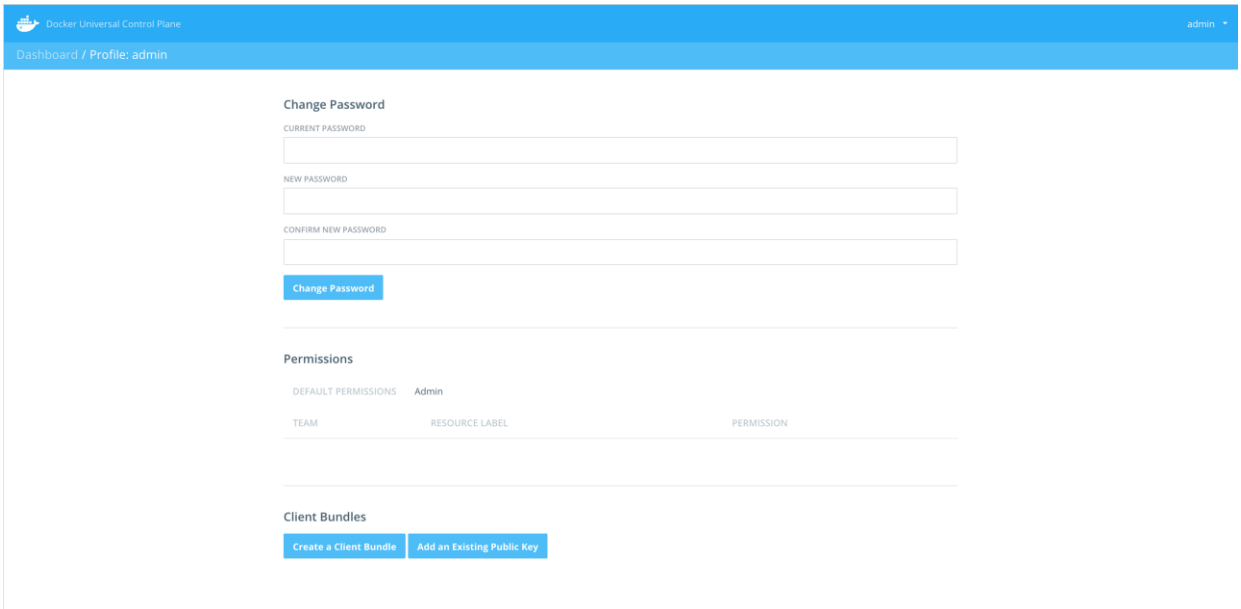
- Docker Toolbox Overview - https://docs.docker.com/toolbox/overview/
- Docker for Mac or Windows - https://www.docker.com/products/docker-toolbox

For newer versions of Mac or Windows use -- https://docs.docker.com/docker-for-mac/install/ or https://docs.docker.com/docker-for-windows/install/

Docker client bundle can be installed on a mac or windows box where Docker Toolbox has been installed. It can also be run on any of the UCP Controller, DTR or UCP nodes, without having to install the Docker Toolbox.

Procedure to install Docker client bundle is simple. Follow these steps to download the client bundle:

1. Download the client bundle from Docker UCP GUI by navigating to admin > Profile > Create a Client Bundle.



2. Unzip the client bundle file into a new folder on the client machine that has the Docker Toolbox installed:

```
-rw-r--r--@   1 rkharya  staff       13748 Nov 24 11:33 ucp-bundle-admin.zip
RKHARYA-M-H07C:~ rkharya$ mkdir ucp-cvd
RKHARYA-M-H07C:~ rkharya$ mv ucp-bundle-admin.zip ucp-cvd/
RKHARYA-M-H07C:~ rkharya$ cd ucp-cvd/
RKHARYA-M-H07C:ucp-cvd rkharya$ ls -ltr
total 32
-rw-r--r--@ 1 rkharya  staff  13748 Nov 24 11:33 ucp-bundle-admin.zip
RKHARYA-M-H07C:ucp-cvd rkharya$ unzip ucp-bundle-admin.zip
Archive:  ucp-bundle-admin.zip
 extracting: ca.pem
 extracting: cert.pem
 extracting: key.pem
 extracting: cert.pub
 extracting: env.sh
 extracting: env.ps1
 extracting: env.cmd
```

3. Set the environment variable by running env.sh.

```
RKHARYA-M-H07C:ucp-cvd rkharya$ eval $(<env.sh)
RKHARYA-M-H07C:ucp-cvd rkharya$ docker info
Containers: 44
Images: 102
Server Version: swarm/1.2.5
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint
Nodes: 10
 DDC-DTR-1.cisco.com: 10.65.122.64:12376
  └ ID: 7TYN:HVC5:XWF1:SH7Z:AYYW:2GZX:MFLK:TG72:QF4E:BLOH:ZOBX:76EJ
  └ Status: Healthy
  └ Containers: 2 (2 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper
  └ UpdatedAt: 2016-11-24T06:04:22Z
  └ ServerVersion: 1.12.3-cs4
 DDC-DTR-2.cisco.com: 10.65.122.65:12376
  └ ID: UFDJ:UFNQ:47IE:MIOF:RTNF:UX66:CKJ4:6M2O:5EUC:CRCN:UA4U:CFYH
  └ Status: Healthy
  └ Containers: 2 (2 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper
  └ UpdatedAt: 2016-11-24T06:04:16Z
  └ ServerVersion: 1.12.3-cs4
 DDC-DTR-3.cisco.com: 10.65.122.66:12376
  └ ID: IUPK:LN4O:PD2N:2TXN:QTDC:2E6L:YQVB:XJV6:WW6W:GI4M:EYO6:KIBE
  └ Status: Healthy
  └ Containers: 2 (2 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper
  └ UpdatedAt: 2016-11-24T06:04:17Z
  └ ServerVersion: 1.12.3-cs4
 UCP-Ctrl-1.cisco.com: 10.65.122.61:12376
  └ ID: 3OPE:S2YE:KLPG:GMDH:XTW6:IBHD:RYFF:2XSE:WKZP:BDTP:KDW4:T7NA
  └ Status: Healthy
  └ Containers: 10 (10 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper
  └ UpdatedAt: 2016-11-24T06:04:20Z
  └ ServerVersion: 1.12.3-cs4
 UCP-Ctrl-2.cisco.com: 10.65.122.62:12376
  └ ID: 7KIA:NZBQ:NMYT:LMYL:CZBD:TWE2:JTHK:45HX:YBWW:KOHC:TEGO:3GQO
  └ Status: Healthy
  └ Containers: 10 (10 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 58
  └ Reserved Memory: 0 B / 131.8 GiB
  └ Labels: kernelversion=3.10.0-327.36.3.el7.x86_64, operatingsystem=Red Hat Enterprise Linux, storagedriver=devicemapper
  └ UpdatedAt: 2016-11-24T06:04:14Z
  └ ServerVersion: 1.12.3-cs4
```

4. Now you are good to use the client, as shown in above example. Setting up environment variable enables you to use your admin credential to connect to your deployed Docker Datacenter. Once the environment is set, you can do all container management operations through Docker shell commands sitting outside the Docker UCP cluster itself.

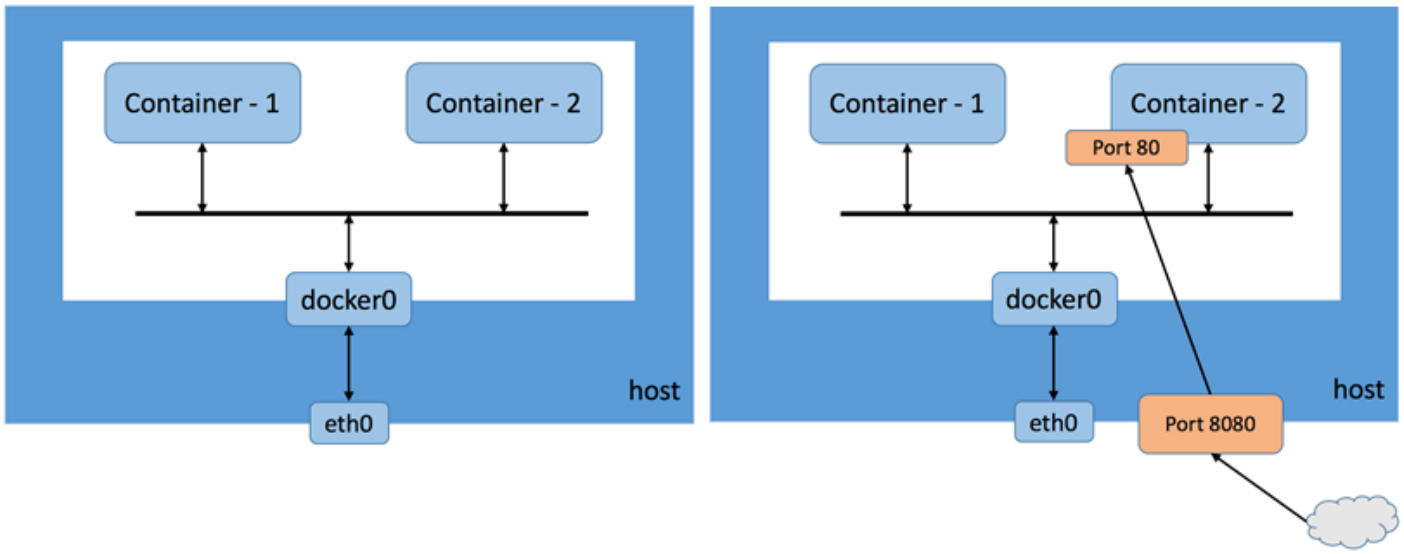
Container Networks

There are various ways in which container-to-container and container-to-host connectivity is provided. This sub-section provides details on the container networking types supported for Docker containers:

- None – None is a simple mode of container networking. In this the deployed container receives a network stack, which lacks an external network interface; but does receive a loopback interface. This mode of container networking is mainly used for testing and staging a container and where container does not need external communications.
- Bridge – A Linux bridge provides a host internal network in which containers on the same host may communicate, but the IP address assigned to them are not accessible from outside of the host. Bridge networking leverages iptables for NAT and port-mapping thus providing single host

networking. Bridge network is a default Docker network type, for example, docker0 is a bridge name where one end of a virtual network interface pair is connected between the bridge and the container. NAT is used to provide communication beyond the host. Bridge networking solves the problems of port-conflict and also solves performance over head associated using NAT. It also provides isolation to containers running on one host.

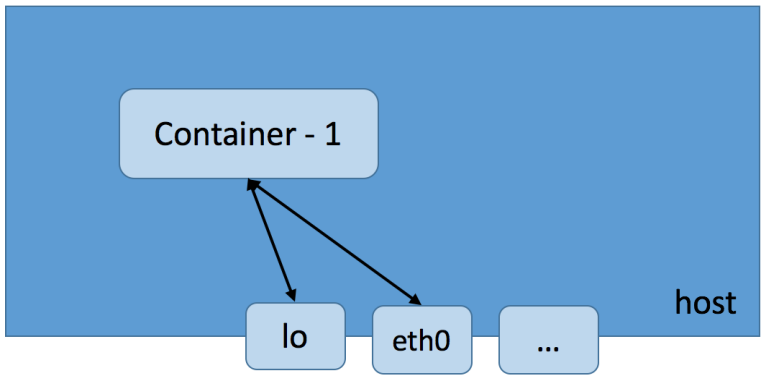
Figure 1. Bridge Network



The first image in the figure shows how a bridge network communicates and the adjacent images shows how a service on Container-2 gets exposed to outside world via host port 8080 mapped to port 80 on the container.

- Host – In this type of network, a newly created container shares its network namespace with the host, providing higher performance and eliminating the need for NAT. The bottle-neck in this type of network is port-conflicts and since container has full-access to hosts interfaces it run into security risks.

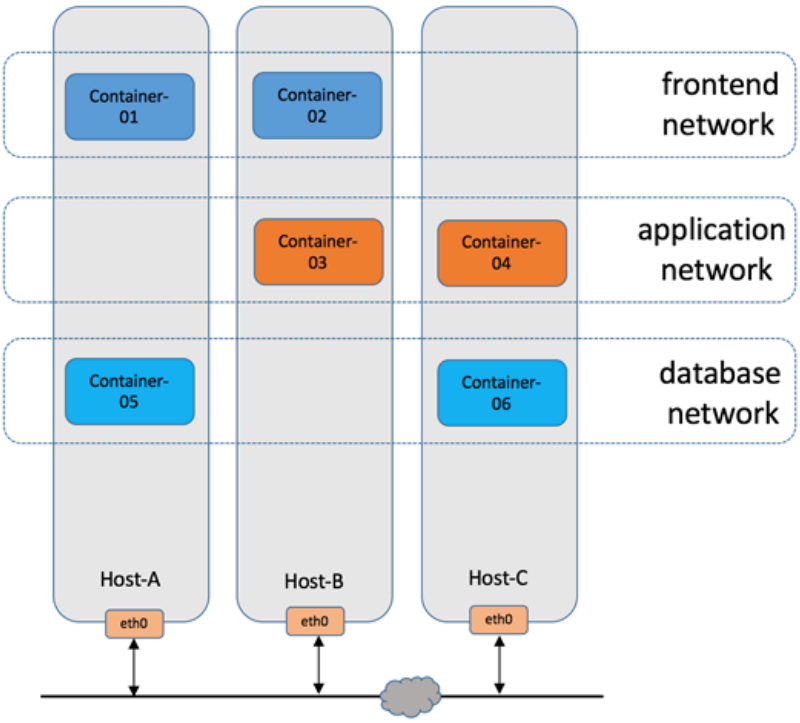
Figure 2. Host Network



- Overlay – It uses networking tunnels to deliver communication across hosts. This allows containers to behave as if they are on the same host by tunneling network subnets from one host to the other. Essentially, it is like spanning one network across multiple hosts. VxLAN encapsulation is the tunneling choice for Docker libnetworks. Multi-host networking requires additional parameters when

launching Docker daemon, as well as a key-value store. Overlay networks focus on the cross-host communication challenge. Containers on the same host that are connected to two different overlay networks cannot communicate with each other via local bridge – they are segmented from one another.

Figure 3. Overlay Network



Deploying container/application container using overlay network

Follow these steps to deploy container/application container using overlay network:

1. Log in to any Linux machine with Docker UCP client bundle installed. To list out existing available networks, run the command ‘docker network ls’ as shown:

```
RKHARYA-M-H07C:ucp-cvd-2.0 rkharya$ docker network ls
NETWORK ID          NAME                                DRIVER              SCOPE
ccf28ac33863        DDC-DTR-1.cisco.com/bridge          bridge              local
750e905d405d        DDC-DTR-1.cisco.com/docker_gwbridge bridge              local
1accb87e11f9        DDC-DTR-1.cisco.com/dtr-br          bridge              local
da9d569ee0c5        DDC-DTR-1.cisco.com/host            host                local
c4693cb2403a        DDC-DTR-1.cisco.com/none            null                local
5d1451a5d068        DDC-DTR-2.cisco.com/bridge          bridge              local
7a50ef817765        DDC-DTR-2.cisco.com/docker_gwbridge bridge              local
fb7b87528697        DDC-DTR-2.cisco.com/dtr-br          bridge              local
bff67bf0a19b        DDC-DTR-2.cisco.com/host            host                local
681048ebded2        DDC-DTR-2.cisco.com/none            null                local
0bbf5442de58        DDC-DTR-3.cisco.com/bridge          bridge              local
8f1509b99615        DDC-DTR-3.cisco.com/docker_gwbridge bridge              local
05092cec0d75        DDC-DTR-3.cisco.com/dtr-br          bridge              local
8d9a7b0221fb        DDC-DTR-3.cisco.com/host            host                local
451a1ac41ba3        DDC-DTR-3.cisco.com/none            null                local
7ab828b762f9        UCP-Ctrl-1.cisco.com/bridge         bridge              local
1a3550f0b15d        UCP-Ctrl-1.cisco.com/docker_gwbridge bridge             local
6d882d0e48ea        UCP-Ctrl-1.cisco.com/host           host                local
b64bc40f5eb4        UCP-Ctrl-1.cisco.com/none           null                local
2aafef5f93a0        UCP-Ctrl-2.cisco.com/bridge         bridge              local
b9be940ea2ab        UCP-Ctrl-2.cisco.com/docker_gwbridge bridge             local
8aa1154da775        UCP-Ctrl-2.cisco.com/host           host                local
fee1d12689e0        UCP-Ctrl-2.cisco.com/none           null                local
cd7f100a8584        UCP-Ctrl-3.cisco.com/bridge         bridge              local
6f8f737b90dc        UCP-Ctrl-3.cisco.com/docker_gwbridge bridge             local
f5b3f8b731c3        UCP-Ctrl-3.cisco.com/host           host                local
c5839dc023aa        UCP-Ctrl-3.cisco.com/none           null                local
8f268d64d70f        UCP-Node-1.cisco.com/bridge         bridge              local
2ff891cb7d29        UCP-Node-1.cisco.com/docker_gwbridge bridge             local
5ad4d0fbea27        UCP-Node-1.cisco.com/host           host                local
177c6a6cd089        UCP-Node-1.cisco.com/none           null                local
3767f8a956ee        UCP-Node-2.cisco.com/bridge         bridge              local
3a1b6963a8d2        UCP-Node-2.cisco.com/docker_gwbridge bridge             local
3226297aef20        UCP-Node-2.cisco.com/host           host                local
2961ad1e3459        UCP-Node-2.cisco.com/none           null                local
fe88846afccc        UCP-Node-3.cisco.com/bridge         bridge              local
afc564de0fa3        UCP-Node-3.cisco.com/docker_gwbridge bridge             local
8a32c0bb7673        UCP-Node-3.cisco.com/host           host                local
9a5a390efea1        UCP-Node-3.cisco.com/none           null                local
906e823b0f5b        UCP-Node-4.cisco.com/bridge         bridge              local
b254f77f4027        UCP-Node-4.cisco.com/docker_gwbridge bridge             local
6f49f7e2d9ec        UCP-Node-4.cisco.com/host           host                local
2748f50beaab        UCP-Node-4.cisco.com/none           null                local
bi3idnflp3g2        dtr-ol                              overlay             swarm
```

2. For creating an overlay network, we need to specify the overlay driver and the network name as shown below:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker network create --driver overlay --subnet 10.100.100.0/24 my-multi-host-network
0a15f44ed575cac1a3faa6149fd573624fdfca6b54db356aa6348c5acf3340be
RKHARYA-M-H07C:ucp-cvd rkharya$ docker network ls
NETWORK ID          NAME                                DRIVER              SCOPE
1780610d2064        DDC-DTR-1.cisco.com/bridge          bridge              local
42c8e959de44        DDC-DTR-1.cisco.com/docker_gwbridge bridge              local
c56364409f7f        DDC-DTR-1.cisco.com/dtr-br          bridge              local
c6fdabbd2410        DDC-DTR-1.cisco.com/host            host                local
912ce0c14663        DDC-DTR-1.cisco.com/none            null                local
7a87bc7fd5cf        DDC-DTR-2.cisco.com/bridge          bridge              local
303200b04022        DDC-DTR-2.cisco.com/docker_gwbridge bridge              local
4940e65a5530        DDC-DTR-2.cisco.com/dtr-br          bridge              local
cdf6cf8fca81        DDC-DTR-2.cisco.com/host            host                local
30fcfa3e8cea        DDC-DTR-2.cisco.com/none            null                local
30dd67b1e42b        DDC-DTR-3.cisco.com/bridge          bridge              local
727ee74b5484        DDC-DTR-3.cisco.com/docker_gwbridge bridge              local
316d04e8cfb6        DDC-DTR-3.cisco.com/dtr-br          bridge              local
260744abbe48        DDC-DTR-3.cisco.com/host            host                local
83b88d056461        DDC-DTR-3.cisco.com/none            null                local
a78c4b683d86        UCP-Ctrl-1.cisco.com/bridge         bridge              local
6d882d0e48ea        UCP-Ctrl-1.cisco.com/host           host                local
b64bc40f5eb4        UCP-Ctrl-1.cisco.com/none           null                local
cce622de6548        UCP-Ctrl-2.cisco.com/bridge         bridge              local
8aa1154da775        UCP-Ctrl-2.cisco.com/host           host                local
fee1d12689e0        UCP-Ctrl-2.cisco.com/none           null                local
43d8251d8932        UCP-Ctrl-3.cisco.com/bridge         bridge              local
f5b3f8b731c3        UCP-Ctrl-3.cisco.com/host           host                local
c5839dc023aa        UCP-Ctrl-3.cisco.com/none           null                local
0916ac7f0cff        UCP-Node-1.cisco.com/bridge         bridge              local
f63bb6b5cf3d        UCP-Node-1.cisco.com/docker_gwbridge bridge             local
e2af781f4d3e        UCP-Node-1.cisco.com/host           host                local
950ecd4379e3        UCP-Node-1.cisco.com/none           null                local
019cdcde6f4c        UCP-Node-2.cisco.com/bridge         bridge              local
19c5b8a8535d        UCP-Node-2.cisco.com/docker_gwbridge bridge             local
7cb1601dd86c        UCP-Node-2.cisco.com/host           host                local
205abad71ea6        UCP-Node-2.cisco.com/none           null                local
8307e2f4f77f        UCP-Node-3.cisco.com/bridge         bridge              local
edd53dbf494e        UCP-Node-3.cisco.com/host           host                local
9591378ad875        UCP-Node-3.cisco.com/none           null                local
836d85d45570        UCP-Node-4.cisco.com/bridge         bridge              local
aa5d97265390        UCP-Node-4.cisco.com/host           host                local
d4bc16afbddb        UCP-Node-4.cisco.com/none           null                local
ea8d33490c75        dtr-ol                              overlay             global
0a15f44ed575        my-multi-host-network               overlay             global
```

The ‘docker network create’ command requires driver, name of the network and the subnet. After creating the network we can verify its successful creation by listing available networks. The last column as shown in

the screenshot above lists the scope of the network created, ‘local’ refers to the network created on the host locally, while ‘global’ scope tells network is created across all the nodes of the cluster.

3. To check the details of the created overlay network, we use ‘docker inspect’ command:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker network inspect my-multi-host-network
[
    {
        "Name": "my-multi-host-network",
        "Id": "0a15f44ed575cac1a3faa6149fd573624fdfca6b54db356aa6348c5acf3340be",
        "Scope": "global",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.100.100.0/24"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

4. Deploy two test ‘busybox’ containers on different hosts using the created overlay network. We are creating two containers for establishing container to container communication:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker run -itd --network=my-multi-host-network --name=OverlayTest1 busybox
7246428e6cee7668aa176c8bf8fa892160bd368b936f6d4e88254488fe2a177d
RKHARYA-M-H07C:ucp-cvd rkharya$ docker run -itd --network=my-multi-host-network --name=OverlayTest2 busybox
f435b85faf043117583cdd1c41dabb74dc1c24c18a41566f7d1e1aeef0074265
RKHARYA-M-H07C:ucp-cvd rkharya$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
7246428e6cee        busybox             "sh"                16 seconds ago      Up 12 seconds                           UCP-Node-3.cisco.
com/OverlayTest1
f435b85faf04        busybox             "sh"                7 seconds ago       Up 3 seconds                            UCP-Node-4.cisco.
com/OverlayTest2
```

5. Let us check the network stack created inside the container to verify that the interface is plumbed correctly and it gets the ip address from the subnet we have specified earlier and the routes it is populated with. We use the command ‘docker exec’ to verify this:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest1 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0A:64:64:02
          inet addr:10.100.100.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe64:6402/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:878 (878.0 B)  TX bytes:648 (648.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2981 (2.9 KiB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest1 netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.18.0.1      0.0.0.0         UG        0 0          0 eth1
10.100.100.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
172.18.0.0      0.0.0.0         255.255.0.0     U         0 0          0 eth1
```

6. From the above screenshot we see that the first container's ip address as 10.100.100.2. We will verify its reachability and make sure that it can ping the external gateway:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest1 ping -c 10 10.65.122.1
PING 10.65.122.1 (10.65.122.1): 56 data bytes
64 bytes from 10.65.122.1: seq=0 ttl=254 time=0.308 ms
64 bytes from 10.65.122.1: seq=1 ttl=254 time=0.331 ms
64 bytes from 10.65.122.1: seq=2 ttl=254 time=0.355 ms
64 bytes from 10.65.122.1: seq=3 ttl=254 time=0.341 ms
64 bytes from 10.65.122.1: seq=4 ttl=254 time=0.335 ms
64 bytes from 10.65.122.1: seq=5 ttl=254 time=0.356 ms
64 bytes from 10.65.122.1: seq=6 ttl=254 time=0.330 ms
64 bytes from 10.65.122.1: seq=7 ttl=254 time=0.322 ms
64 bytes from 10.65.122.1: seq=8 ttl=254 time=0.354 ms
64 bytes from 10.65.122.1: seq=9 ttl=254 time=0.309 ms

--- 10.65.122.1 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.308/0.334/0.356 ms
RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest2 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0A:64:64:03
          inet addr:10.100.100.3  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe64:6403/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1038 (1.0 KiB)  TX bytes:648 (648.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3200 (3.1 KiB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

7. From the above screenshot we see that the first container's ip address as 10.100.100.3. We will now verify if the container-1 and container-2 are pinging.

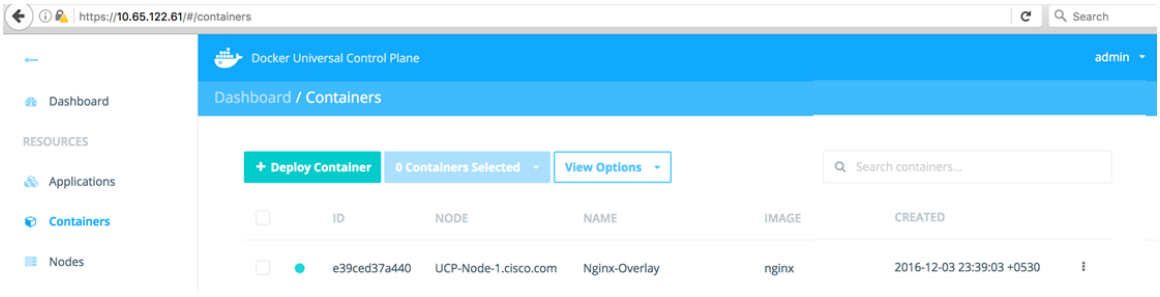
```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest1 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0A:64:64:02
          inet addr:10.100.100.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe64:6402/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:878 (878.0 B)  TX bytes:648 (648.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:36 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4045 (3.9 KiB)  TX bytes:1712 (1.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

RKHARYA-M-H07C:ucp-cvd rkharya$ docker exec OverlayTest1 ping -c 10 10.100.100.3
PING 10.100.100.3 (10.100.100.3): 56 data bytes
64 bytes from 10.100.100.3: seq=0 ttl=64 time=0.613 ms
64 bytes from 10.100.100.3: seq=1 ttl=64 time=0.244 ms
64 bytes from 10.100.100.3: seq=2 ttl=64 time=0.193 ms
64 bytes from 10.100.100.3: seq=3 ttl=64 time=0.267 ms
64 bytes from 10.100.100.3: seq=4 ttl=64 time=0.165 ms
64 bytes from 10.100.100.3: seq=5 ttl=64 time=0.204 ms
64 bytes from 10.100.100.3: seq=6 ttl=64 time=0.169 ms
64 bytes from 10.100.100.3: seq=7 ttl=64 time=0.231 ms
64 bytes from 10.100.100.3: seq=8 ttl=64 time=0.238 ms
64 bytes from 10.100.100.3: seq=9 ttl=64 time=0.207 ms

--- 10.100.100.3 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.165/0.253/0.613 ms
```

8. Deploy an application container for example ‘nginx’ with port-mapping done for httpd services, in order to access its web-service from the upstream network:

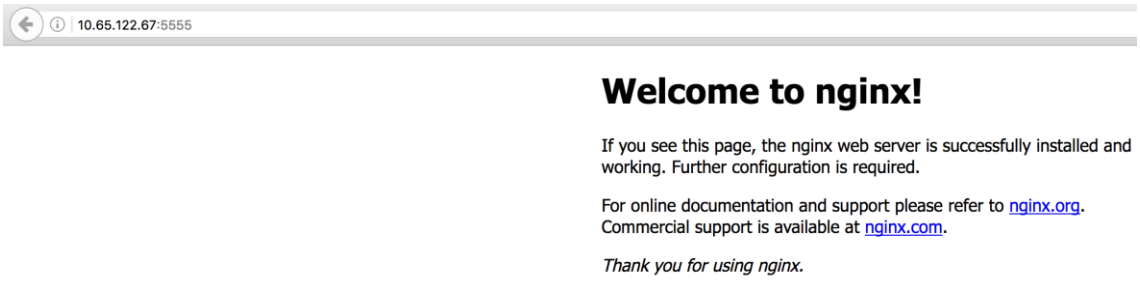
```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker run -itd --network=my-multi-host-network --name=Nginx-Overlay -p 5555:80 nginx
e39ced37a440de538958af330591c82aa209a5578fdc0b1e947406f66d2d6446
RKHARYA-M-H07C:ucp-cvd rkharya$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                  NAMES
e39ced37a440        nginx               "nginx -g 'daemon off"   9 seconds ago       Up 3 seconds        443/tcp, 10.65.122.67:5555->80/tcp     UCP-Node-1.cisco.
com/Nginx-Overlay
7246428e6cee        busybox             "sh"                     38 minutes ago      Up 38 minutes                                              UCP-Node-3.cisco.
com/OverlayTest1
f435b85faf04        busybox             "sh"                     38 minutes ago      Up 38 minutes                                              UCP-Node-4.cisco.
com/OverlayTest2
```

We have mapped the port 5555 on the host to port 80 using ‘-p’ port-mapper switch to deploy an ‘nginx’ container. We can see its deployed on our first UCP-Node-1 host.

9. Open https://<UCP Primary Node ip-address> in a web browser to login to the Docker UCP. The dashboard shows the ‘nginx’ container being deployed:



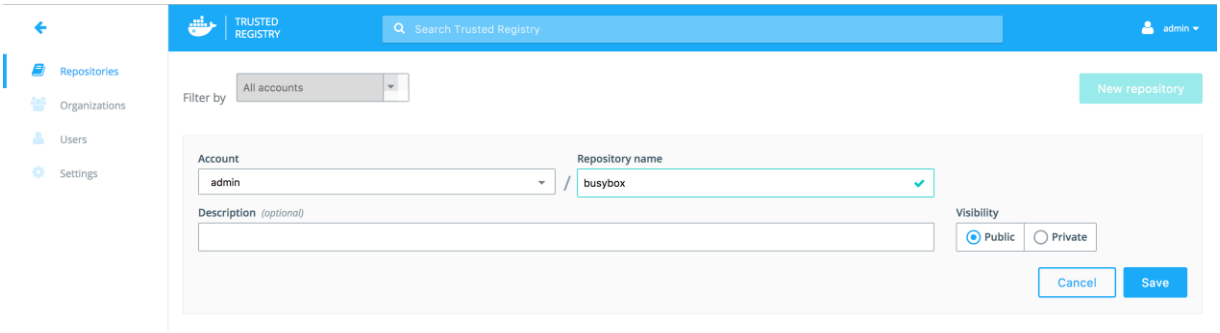
10. Open <node ip where container get deployed>:5555 in a web browser. With the port-mapping in place, we can reach our nginx web-server from outside world, by accessing the port#5555 as shown below:



DTR Operations

This section details about validating the Docker Trusted Registry workflow. As mentioned earlier, DTR provides a way to create a repository of Docker images used for container deployment locally for the enterprise. This eliminates the need for accessing external repos in public domain for downloading images for the containers. This option provides a better control on deploying images for the enterprise and secures application container environment.

1. Login to the DTR GUI and create a repository. In this example we have chosen ‘busybox’ as the repository:



2. Download the sample Docker image (busybox) to the local host. Use ‘docker pull’ command to download the image from the Docker public repo:

```
RKHARYA-M-H07C:ucp-cvd-2.0 rkharya$ docker pull busybox
Using default tag: latest
UCP-Ctrl-3.cisco.com: Pulling busybox:latest... : downloaded
UCP-Node-4.cisco.com: Pulling busybox:latest... : downloaded
UCP-Ctrl-1.cisco.com: Pulling busybox:latest... : downloaded
UCP-Node-3.cisco.com: Pulling busybox:latest... : downloaded
UCP-Ctrl-2.cisco.com: Pulling busybox:latest... : downloaded
DDC-DTR-2.cisco.com: Pulling busybox:latest... : downloaded
UCP-Node-1.cisco.com: Pulling busybox:latest... : downloaded
DDC-DTR-1.cisco.com: Pulling busybox:latest... : downloaded
UCP-Node-2.cisco.com: Pulling busybox:latest... : downloaded
DDC-DTR-3.cisco.com: Pulling busybox:latest... : downloaded
```

The downloaded image gets pulled to all our cluster nodes as shown in the screenshot above. This enables us to run the container using this image on any of the host.

3. To list all the images that we have on the setup use ‘docker image’ command:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
docker/dtr              latest              sha256:4aad0        2 weeks ago         100.1 MB
docker/dtr              <none>              sha256:4aad0        2 weeks ago         100.1 MB
busybox                 latest              sha256:e02e8        7 weeks ago         1.093 MB
busybox                 <none>              sha256:e02e8        7 weeks ago         1.093 MB
docker/ucp-controller   1.1.4               sha256:75c97        8 weeks ago         25.82 MB
docker/ucp-controller   <none>              sha256:75c97        8 weeks ago         25.82 MB
docker/ucp-proxy        1.1.4               sha256:ee0c1        8 weeks ago         16.88 MB
docker/ucp-proxy        <none>              sha256:ee0c1        8 weeks ago         16.88 MB
docker/ucp-etcd         1.1.4               sha256:6f46c        8 weeks ago         37.71 MB
docker/ucp-etcd         <none>              sha256:6f46c        8 weeks ago         37.71 MB
docker/ucp-dsinfo       1.1.4               sha256:b7ad8        8 weeks ago         46.39 MB
docker/ucp-dsinfo       <none>              sha256:b7ad8        8 weeks ago         46.39 MB
docker/ucp-cfssl        1.1.4               sha256:df495        8 weeks ago         54.67 MB
docker/ucp-cfssl        <none>              sha256:df495        8 weeks ago         54.67 MB
docker/ucp-auth-store   1.1.4               sha256:b1649        8 weeks ago         58.65 MB
docker/ucp-auth-store   <none>              sha256:b1649        8 weeks ago         58.65 MB
docker/ucp              1.1.4               sha256:08cd6        8 weeks ago         20.03 MB
docker/ucp              <none>              sha256:08cd6        8 weeks ago         20.03 MB
docker/ucp-auth         1.1.4               sha256:150b6        8 weeks ago         26.3 MB
docker/ucp-auth         <none>              sha256:150b6        8 weeks ago         26.3 MB
docker/ucp-swarm        1.1.4               sha256:fa1c9        3 months ago        19.47 MB
docker/ucp-swarm        <none>              sha256:fa1c9        3 months ago        19.47 MB
docker/ucp-compose      1.1.4               sha256:15b7b        3 months ago        58.6 MB
docker/ucp-compose      <none>              sha256:15b7b        3 months ago        58.6 MB
docker/dtr              2.0.3               sha256:c69ba        3 months ago        51.51 MB
docker/dtr              <none>              sha256:c69ba        3 months ago        51.51 MB
docker/dtr-registry     2.0.3               sha256:ed435        3 months ago        110.6 MB
docker/dtr-registry     <none>              sha256:ed435        3 months ago        110.6 MB
docker/dtr-rethink      2.0.3               sha256:0a914        3 months ago        79.95 MB
docker/dtr-rethink      <none>              sha256:0a914        3 months ago        79.95 MB
docker/dtr-nginx        2.0.3               sha256:645bd        3 months ago        89.83 MB
docker/dtr-nginx        <none>              sha256:645bd        3 months ago        89.83 MB
docker/dtr-api          2.0.3               sha256:f7bce        3 months ago        94.6 MB
docker/dtr-api          <none>              sha256:f7bce        3 months ago        94.6 MB
docker/dtr-etcd         v2.2.4              sha256:205bc        10 months ago       28.65 MB
docker/dtr-etcd         <none>              sha256:205bc        10 months ago       28.65 MB
RKHARYA-M-H07C:ucp-cvd rkharya$
```

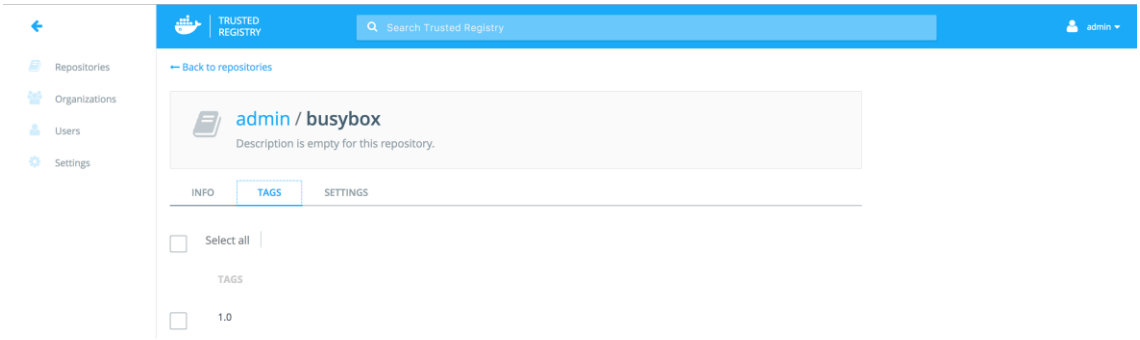
4. Tag the image with version that we want to put in our DTR repo. Tags help in version control. So any dev changes to this image version will have an incremental version number and will be pushed to the DTR repo. This way we have an option to quickly upgrade and/or downgrade to the required version.

```
RKHARYA-M-H07C:ucp-cvd rkharya$  docker tag busybox 10.65.122.64/admin/busybox:1.0
RKHARYA-M-H07C:ucp-cvd rkharya$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
docker/dtr                    latest              sha256:4aad0        2 weeks ago         100.1 MB
docker/dtr                    <none>              sha256:4aad0        2 weeks ago         100.1 MB
10.65.122.64/admin/busybox    1.0                 sha256:e02e8        7 weeks ago         1.093 MB
busybox                       latest              sha256:e02e8        7 weeks ago         1.093 MB
busybox                       <none>              sha256:e02e8        7 weeks ago         1.093 MB
docker/ucp-controller         1.1.4               sha256:75c97        8 weeks ago         25.82 MB
docker/ucp-controller         <none>              sha256:75c97        8 weeks ago         25.82 MB
docker/ucp-proxy              1.1.4               sha256:ee0c1        8 weeks ago         16.88 MB
docker/ucp-proxy              <none>              sha256:ee0c1        8 weeks ago         16.88 MB
docker/ucp-etcd               1.1.4               sha256:6f46c        8 weeks ago         37.71 MB
docker/ucp-etcd               <none>              sha256:6f46c        8 weeks ago         37.71 MB
docker/ucp-dsinfo             1.1.4               sha256:b7ad8        8 weeks ago         46.39 MB
docker/ucp-dsinfo             <none>              sha256:b7ad8        8 weeks ago         46.39 MB
docker/ucp-cfssl              1.1.4               sha256:df495        8 weeks ago         54.67 MB
docker/ucp-cfssl              <none>              sha256:df495        8 weeks ago         54.67 MB
docker/ucp-auth-store         1.1.4               sha256:b1649        8 weeks ago         58.65 MB
docker/ucp-auth-store         <none>              sha256:b1649        8 weeks ago         58.65 MB
docker/ucp                    1.1.4               sha256:08cd6        8 weeks ago         20.03 MB
docker/ucp                    <none>              sha256:08cd6        8 weeks ago         20.03 MB
docker/ucp-auth               1.1.4               sha256:150b6        8 weeks ago         26.3 MB
docker/ucp-auth               <none>              sha256:150b6        8 weeks ago         26.3 MB
docker/ucp-swarm              1.1.4               sha256:fa1c9        3 months ago        19.47 MB
docker/ucp-swarm              <none>              sha256:fa1c9        3 months ago        19.47 MB
docker/ucp-compose            1.1.4               sha256:15b7b        3 months ago        58.6 MB
docker/ucp-compose            <none>              sha256:15b7b        3 months ago        58.6 MB
docker/dtr                    2.0.3               sha256:c69ba        3 months ago        51.51 MB
docker/dtr                    <none>              sha256:c69ba        3 months ago        51.51 MB
docker/dtr-registry           2.0.3               sha256:ed435        3 months ago        110.6 MB
docker/dtr-registry           <none>              sha256:ed435        3 months ago        110.6 MB
docker/dtr-rethink            2.0.3               sha256:0a914        3 months ago        79.95 MB
docker/dtr-rethink            <none>              sha256:0a914        3 months ago        79.95 MB
docker/dtr-nginx              2.0.3               sha256:645bd        3 months ago        89.83 MB
docker/dtr-nginx              <none>              sha256:645bd        3 months ago        89.83 MB
docker/dtr-api                2.0.3               sha256:f7bce        3 months ago        94.6 MB
docker/dtr-api                <none>              sha256:f7bce        3 months ago        94.6 MB
docker/dtr-etcd               v2.2.4              sha256:205bc        10 months ago       28.65 MB
docker/dtr-etcd               <none>              sha256:205bc        10 months ago       28.65 MB
```

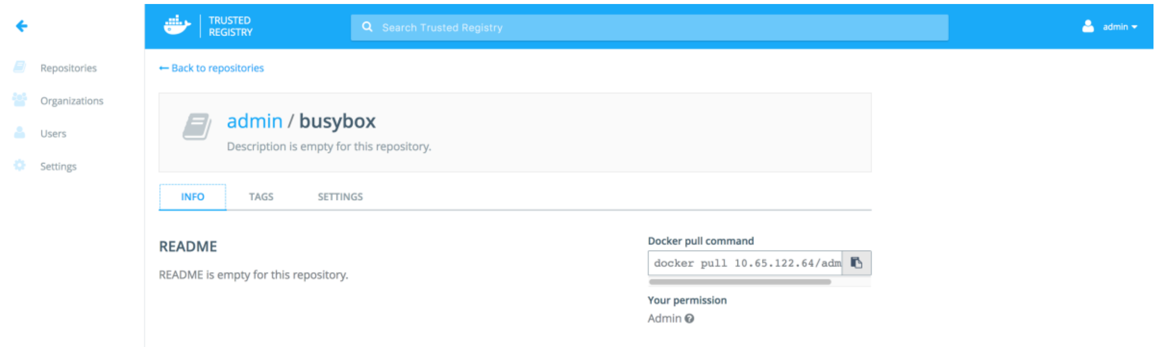
5. To push our tagged image to local DTR repository we need to log-in to the local DTR and push the image as shown below:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker login 10.65.122.64
Username (admin): admin
WARNING: login credentials saved in /Users/rkharya/.docker/config.json
Login Succeeded
RKHARYA-M-H07C:ucp-cvd rkharya$ docker push 10.65.122.64/admin/busybox:1.0
The push refers to a repository [10.65.122.64/admin/busybox]
e88b3f82283b: Pushed
1.0: digest: sha256:7a5001c4aa6ea80576bf05fdad590be298b1605ab43472cd8a1dce9f8265f365 size: 2092
RKHARYA-M-H07C:ucp-cvd rkharya$
```

6. Verify that the image was successfully pushed from the DTR dashboard:



7. Select 1.0 tag in the TAGS tab and press INFO tab to get the command to pull this image from the local repo for deployment:



8. Now validate the container deployment using local DTR image repo:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker run -it --name test-busybox 10.65.122.64/admin/busybox:1.0
/ # uname -a
Linux 7fd6f0ce6fdf 3.10.0-327.36.3.el7.x86_64 #1 SMP Thu Oct 20 04:56:07 EDT 2016 x86_64 GNU/Linux
/ # ifconfig -a
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:04
          inet addr:172.17.0.4  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:648 (648.0 B)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # ping 10.65.122.1
PING 10.65.122.1 (10.65.122.1): 56 data bytes
64 bytes from 10.65.122.1: seq=0 ttl=254 time=66.421 ms
64 bytes from 10.65.122.1: seq=1 ttl=254 time=0.340 ms
64 bytes from 10.65.122.1: seq=2 ttl=254 time=0.375 ms
^C
--- 10.65.122.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.340/22.378/66.421 ms
/ #
```

9. The complete URL of the busybox image provides the path for the ‘docker run’ command from where the image for the container deployment can be taken. Note that we have already logged in to our local DTR before using ‘docker login’ command; this facilitated us to get connected on the fly, use the image from there and deploy a container from it. The command ‘docker ps’ lists all the containers.

```
/ # RKHARYA-M-H07C:ucp-cvd rkharya$ docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED              STATUS              PORTS                                                  NAMES
7fd6f0ce6fdf        10.65.122.64/admin/busybox:1.0     "sh"                     About a minute ago   Up About a minute                                                          UCP-Node-3.
cisco.com/test-busybox
95e8f78659ba        docker/dtr-api:2.0.3               "/bin/server"            5 days ago           Up 41 hours         80/tcp                                                 DDC-DTR-3.c
isco.com/dtr-api-448e20d5d8df
f82b6c8d3f47        docker/dtr-registry:2.0.3          "/init --skip-runit /"   5 days ago           Up 41 hours         3000/tcp                                               DDC-DTR-3.c
isco.com/dtr-registry-448e20d5d8df
a8cd135fa234        docker/dtr-rethink:2.0.3           "/start.sh"              5 days ago           Up 41 hours         8080/tcp                                               DDC-DTR-3.c
isco.com/dtr-rethinkdb-448e20d5d8df
39cf0379dd70        sha256:205bc                       "/etcd"                  5 days ago           Up 41 hours         2379-2380/tcp, 4001/tcp, 7001/tcp                      DDC-DTR-3.c
isco.com/dtr-etcd-448e20d5d8df
94ad6d6860c2        docker/dtr-nginx:2.0.3             "/init --skip-runit /"   5 days ago           Up 39 hours         10.65.122.65:80->80/tcp, 10.65.122.65:443->443/tcp     DDC-DTR-2.c
isco.com/dtr-nginx-cc57c8974d24
1daa74c82d28        docker/dtr-api:2.0.3               "/bin/server"            5 days ago           Up 39 hours         80/tcp                                                 DDC-DTR-2.c
isco.com/dtr-api-cc57c8974d24
9c49956dfb18        docker/dtr-registry:2.0.3          "/init --skip-runit /"   5 days ago           Up 39 hours         3000/tcp                                               DDC-DTR-2.c
isco.com/dtr-registry-cc57c8974d24
7faaf86ec68f        docker/dtr-rethink:2.0.3           "/start.sh"              5 days ago           Up 39 hours         8080/tcp                                               DDC-DTR-2.c
isco.com/dtr-rethinkdb-cc57c8974d24
3cb0fd285daa        sha256:205bc                       "/etcd"                  5 days ago           Up 39 hours         2379-2380/tcp, 4001/tcp, 7001/tcp                      DDC-DTR-2.c
isco.com/dtr-etcd-cc57c8974d24
c58b33ba5a29        docker/dtr-nginx:2.0.3             "/init --skip-runit /"   5 days ago           Up 39 hours         10.65.122.64:80->80/tcp, 10.65.122.64:443->443/tcp     DDC-DTR-1.c
isco.com/dtr-nginx-ea5f0db3c5fb
fbdc7a59191f        docker/dtr-api:2.0.3               "/bin/server"            5 days ago           Up 39 hours         80/tcp                                                 DDC-DTR-1.c
isco.com/dtr-api-ea5f0db3c5fb
cfc97b639ac7        docker/dtr-registry:2.0.3          "/init --skip-runit /"   5 days ago           Up 39 hours         3000/tcp                                               DDC-DTR-1.c
isco.com/dtr-registry-ea5f0db3c5fb
90d462424b8f        docker/dtr-rethink:2.0.3           "/start.sh"              5 days ago           Up 39 hours         8080/tcp                                               DDC-DTR-1.c
isco.com/dtr-rethinkdb-ea5f0db3c5fb
17cf28020a39        sha256:205bc                       "/etcd"                  5 days ago           Up 39 hours         2379-2380/tcp, 4001/tcp, 7001/tcp                      DDC-DTR-1.c
isco.com/dtr-etcd-ea5f0db3c5fb
RKHARYA-M-H07C:ucp-cvd rkharya$
```

Sample WordPress Application Container Deployment

Docker provides an automated way to deploy entire application stack at one go and scale with ease and simplicity. Docker Compose is another tool for such an application deployment use case. The tool needs a yml file, detailing all the configuration parameters for the stack, including network and volume storage. When we run 'docker compose' using the configuration file we get the whole application stack up in just a few minutes. And these applications are running inside the containers that were deployed.

In this section, we are going to deploy a two-tier container application called WordPress, which is a popular blogging website world-wide. The application stack has a front-end WordPress and back-end database mariadb.

1. Create a yml file for Docker Compose as shown in the screenshot:

```
RKHARYA-M-H07C:ucp-cvd rkharya$ cat docker-compose.yml
wordpress:
  image: wordpress
  links:
    - db:mysql
  ports:
    - 8080:80

db:
  image: mariadb
  environment:
    MYSQL_ROOT_PASSWORD: Nbv12345!
```

2. Create a directory name 'wordpress' and put the yml file into it. This yml file is taken by default when running docker compose:

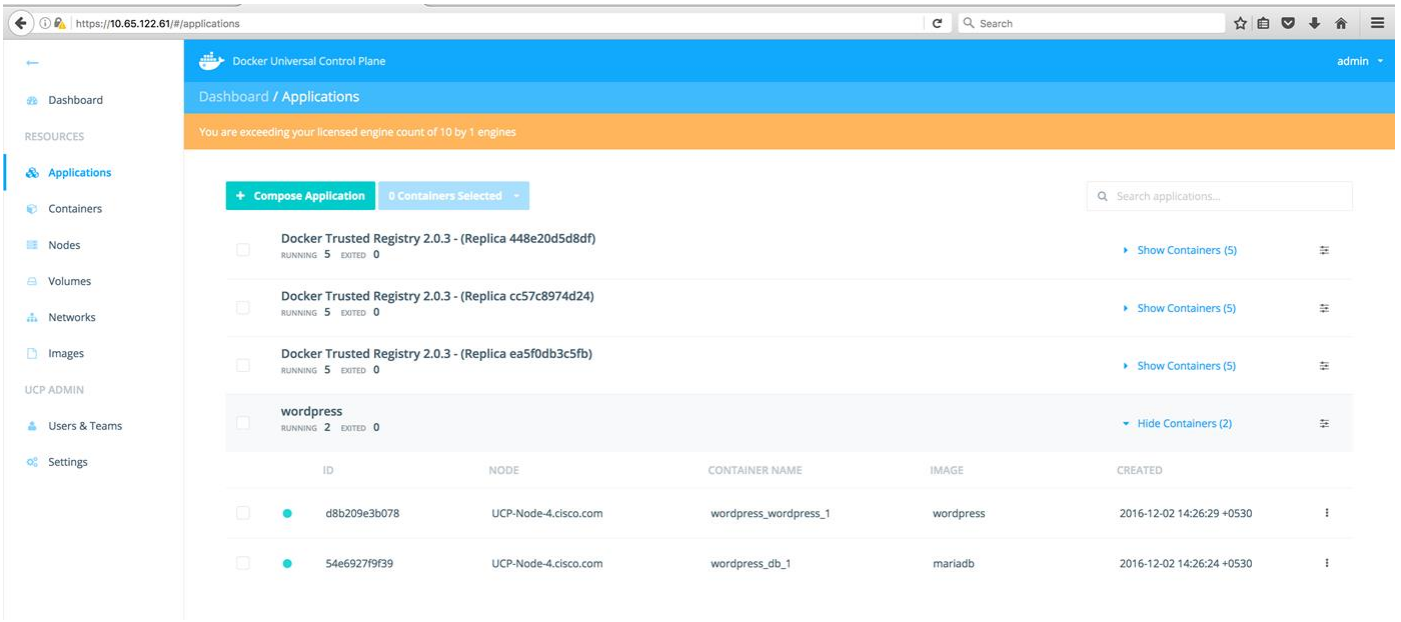
```
RKHARYA-M-H07C:ucp-cvd rkharya$ docker-compose up -d
Pulling db (mariadb:latest)...
DDC-DTR-2.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Ctrl-2.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Node-1.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Ctrl-3.cisco.com: Pulling mariadb:latest... : downloaded
DDC-DTR-3.cisco.com: Pulling mariadb:latest... : downloaded
DDC-DTR-1.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Node-2.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Node-3.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Ctrl-1.cisco.com: Pulling mariadb:latest... : downloaded
UCP-Node-4.cisco.com: Pulling mariadb:latest... : downloaded
Pulling wordpress (wordpress:latest)...
UCP-Node-3.cisco.com: Pulling wordpress:latest... : downloaded
DDC-DTR-1.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Ctrl-3.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Node-4.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Ctrl-2.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Node-2.cisco.com: Pulling wordpress:latest... : downloaded
DDC-DTR-3.cisco.com: Pulling wordpress:latest... : downloaded
DDC-DTR-2.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Node-1.cisco.com: Pulling wordpress:latest... : downloaded
UCP-Ctrl-1.cisco.com: Pulling wordpress:latest... : downloaded
Creating ucpcvd_db_1
Creating ucpcvd_wordpress_1
```

So this way application containers gets pulled from the Docker hub and gets installed.

3. The status of wordpress application containers is shown below:

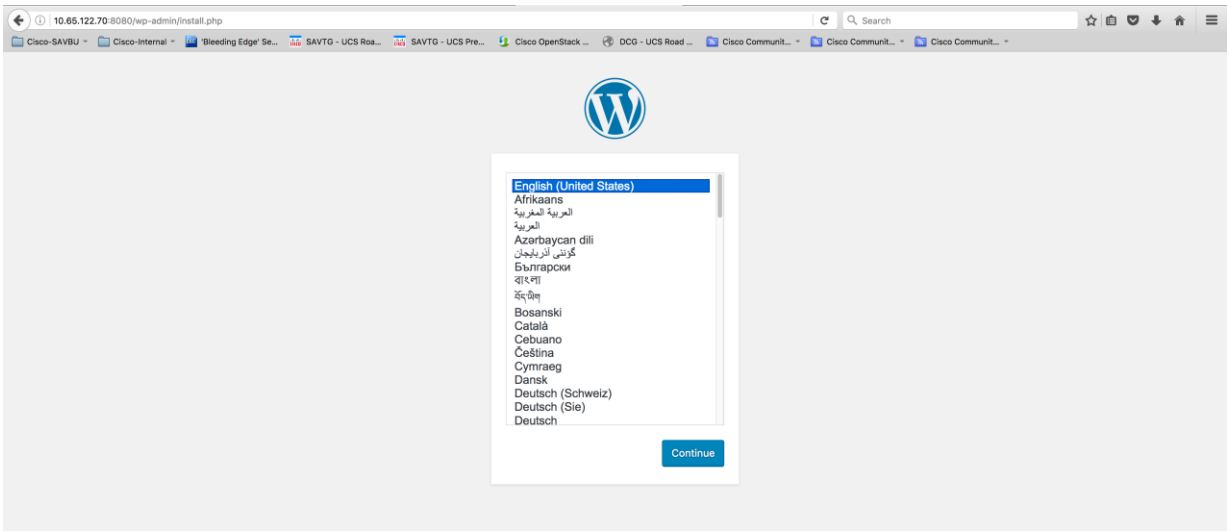
```
RKHARYA-M-H07C:wordpress rkharya$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
d8b209e3b078        wordpress           "docker-entrypoint.sh"   3 minutes ago       Up 3 minutes        10.65.122.70:8080->80/tcp   UCP-Node-4.cisco.
com/wordpress_wordpress_1
54e6927f9f39        mariadb             "docker-entrypoint.sh"   3 minutes ago       Up 3 minutes        3306/tcp                  UCP-Node-4.cisco.
```

4. You can see the wordpress status on Docker UCP dashboard:

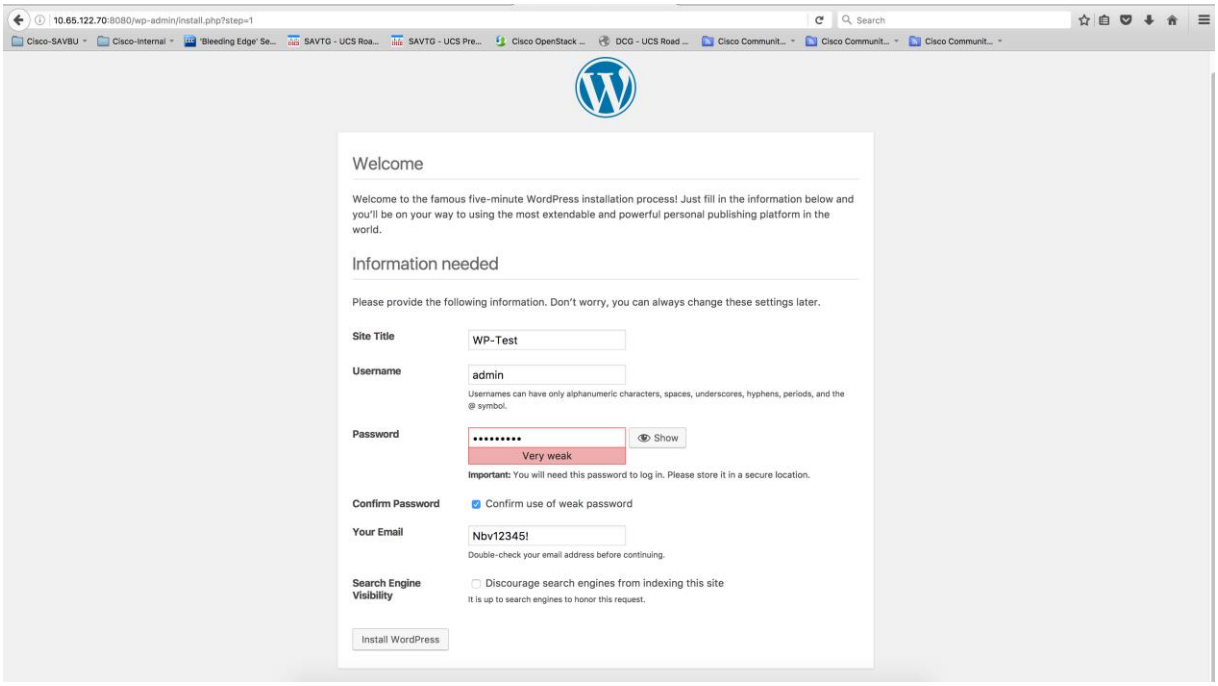


5. The screenshots below shows the initial WordPress application configuration pages:

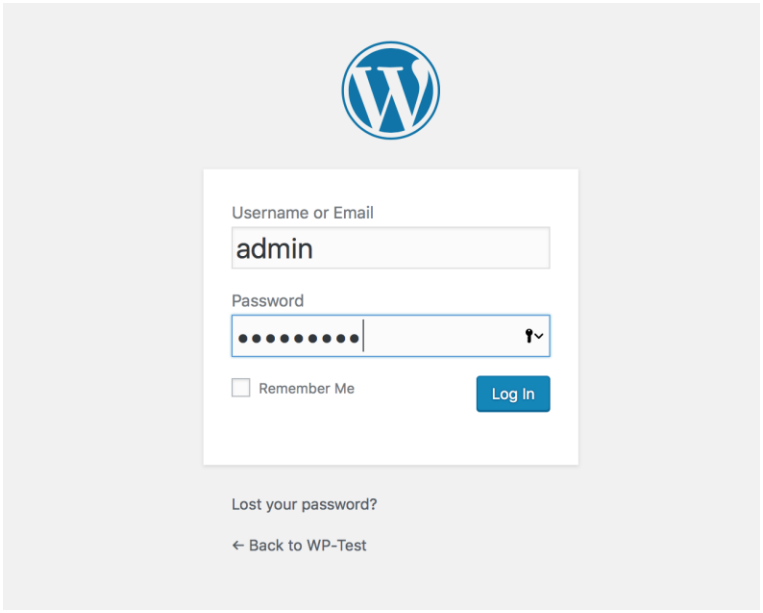
   a. Select the language. Click Continue.



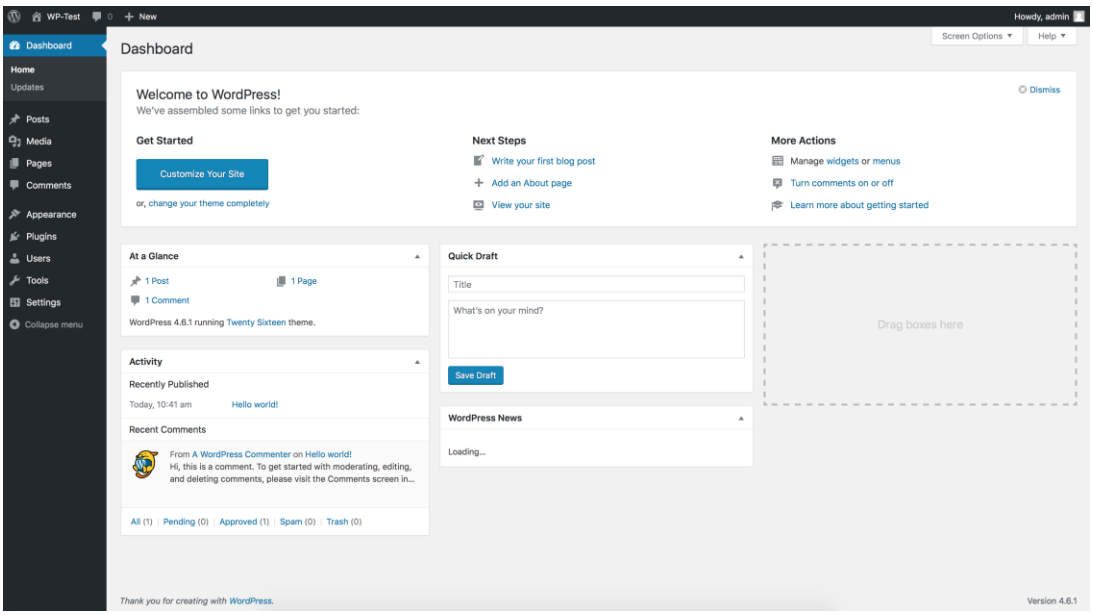
   b. Enter the user credentials and other details as needed.



c. Login to the WordPress application.



d. After the successful login, you will see the WordPress dashboard.



High-Availability Tests

This solution is designed to provide service availability for running the container applications in the event of a failure at the hardware and/or software stack. Performance impact is expected in the service degraded state. To test these scenarios and their impact on system performance, faults were injected on an up and running setup. Docker UCP Controller, UCP and DTR nodes were subjected to the various failure scenarios and their performance was observed. Node failure was simulated by rebooting and shutting of one node at a time and removal of the node from the chassis.

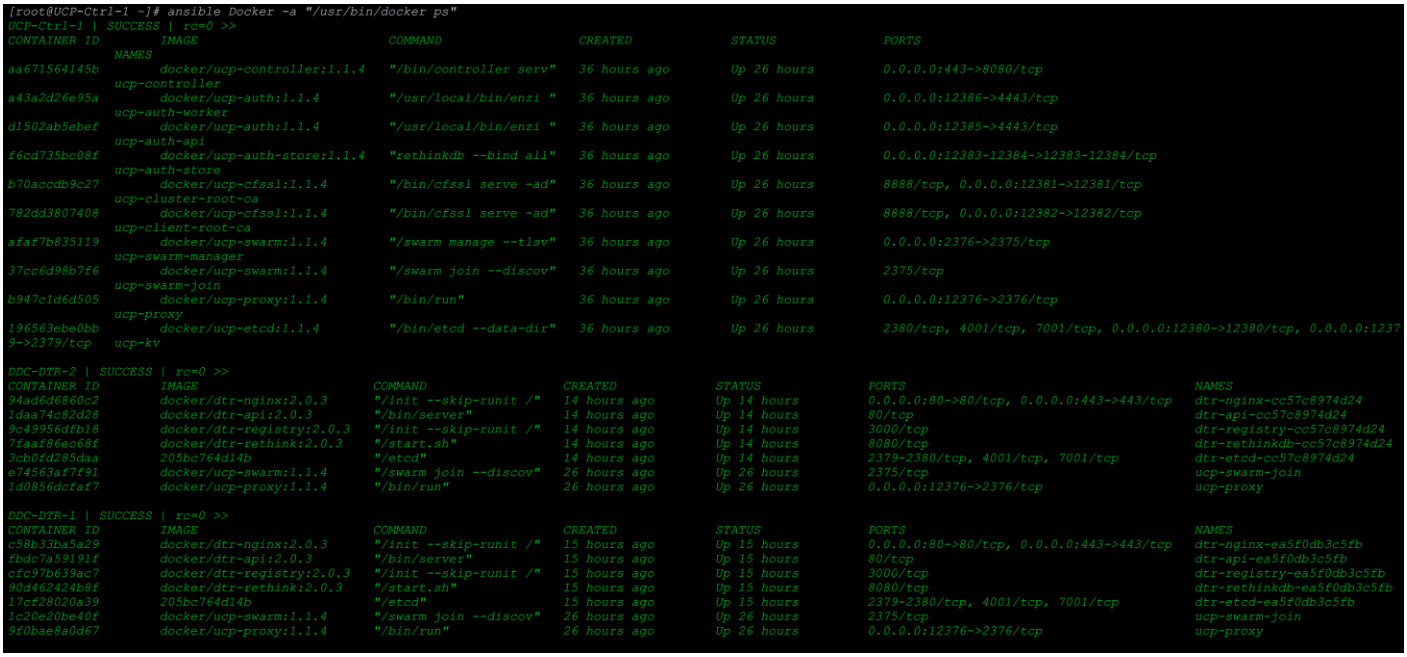
Tests Performed on Docker Datacenter

1. UCP Node reboot/shutdown:
   - Tests passed with all the running containers coming up online without any delay.
2. DTR Node reboot:
   - Tests passed with DTR application containers coming up online without any delay.
3. UCP Controller Master/Replica node reboot:
   - Tests passed. UCP management control plane through CLI/GUI was available for cluster/application container administration. Controller node reboot did not impact container data-path and subsequent re-convergence of the cluster.
4. UCP Controller Master/Replica node shutdown:
   - Tests passed. Controller node shutdown does not impact cluster and application container management operations and associated data path. All such activities remain unaffected and were serviced by surviving controller nodes. Cluster re-convergence succeeded on shutdown node bring-up.
   - Containers which were already deployed and the applications running were seen not affected on the UCP nodes during the unavailability of that UCP controller node.
5. Docker Engine Service Restart on UCP nodes:

- Tests passed with the infrastructure containers and deployed application containers coming up online without any delay.

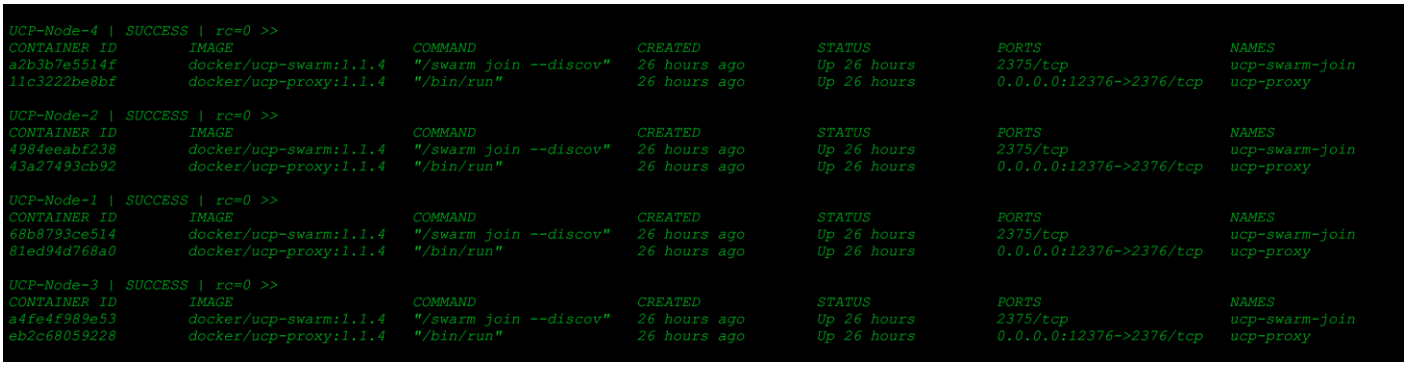
6. Docker Engine Service Restart on DTR nodes:
   - Tests passed with DTR application containers coming up without any delay.
7. Docker Engine Service Restart on Controller Master/Replica Nodes:
   - Tests passed; however, there was a delay seen in restoring the services.
   - Restoration time was similar to the UCP Controller Master/Replica node reboot (#3) above.
8. Infrastructure component level service/process restart:
   - Docker Datacenter services runs on cluster nodes as infrastructure containers except for the Docker Engine, which is a 'systemd' process controlled at the operating system level through 'systemctl' control calls. Docker Engine restart tests have been covered under #7 above. As part of this test case, we identified a few key services and components of the software stack for each node types.

   a. Controller and DTR nodes have the following infrastructure containers running:



```
[root@UCP-Ctrl-1 ~]# ansible Docker -a "/usr/bin/docker ps"
UCP-Ctrl-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS
          NAMES
aa671564145b        docker/ucp-controller:1.1.4    "/bin/controller serv"   36 hours ago        Up 26 hours         0.0.0.0:443->8080/tcp
          ucp-controller
a43a2d26e95a        docker/ucp-auth:1.1.4          "/usr/local/bin/enzi "   36 hours ago        Up 26 hours         0.0.0.0:12386->4443/tcp
          ucp-auth-worker
d1502ab5ebef        docker/ucp-auth:1.1.4          "/usr/local/bin/enzi "   36 hours ago        Up 26 hours         0.0.0.0:12385->4443/tcp
          ucp-auth-api
f6cd735bc08f        docker/ucp-auth-store:1.1.4    "rethinkdb --bind all"   36 hours ago        Up 26 hours         0.0.0.0:12383-12384->12383-12384/tcp
          ucp-auth-store
b70accdb9c27        docker/ucp-cfssl:1.1.4         "/bin/cfssl serve -ad"   36 hours ago        Up 26 hours         8888/tcp, 0.0.0.0:12381->12381/tcp
          ucp-cluster-root-ca
782dd3807408        docker/ucp-cfssl:1.1.4         "/bin/cfssl serve -ad"   36 hours ago        Up 26 hours         8888/tcp, 0.0.0.0:12382->12382/tcp
          ucp-client-root-ca
afaf7b835119        docker/ucp-swarm:1.1.4         "/swarm manage --tlsv"   36 hours ago        Up 26 hours         0.0.0.0:2376->2375/tcp
          ucp-swarm-manager
37cc6d98b7f6        docker/ucp-swarm:1.1.4         "/swarm join --discov"   36 hours ago        Up 26 hours         2375/tcp
          ucp-swarm-join
b947c1d6d505        docker/ucp-proxy:1.1.4         "/bin/run"               36 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp
          ucp-proxy
196563ebe0bb        docker/ucp-etcd:1.1.4          "/bin/etcd --data-dir"   36 hours ago        Up 26 hours         2380/tcp, 4001/tcp, 7001/tcp, 0.0.0.0:12380->12380/tcp, 0.0.0.0:1237
9->2379/tcp   ucp-kv

DDC-DTR-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                                          NAMES
94ad6d6860c2        docker/dtr-nginx:2.0.3         "/init --skip-runit /"   14 hours ago        Up 14 hours         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp       dtr-nginx-cc57c8974d24
1daa74c82d28        docker/dtr-api:2.0.3           "/bin/server"            14 hours ago        Up 14 hours         80/tcp                                         dtr-api-cc57c8974d24
9c49956dfb18        docker/dtr-registry:2.0.3      "/init --skip-runit /"   14 hours ago        Up 14 hours         3000/tcp                                       dtr-registry-cc57c8974d24
7faaf86ec68f        docker/dtr-rethink:2.0.3       "/start.sh"              14 hours ago        Up 14 hours         8080/tcp                                       dtr-rethinkdb-cc57c8974d24
3cb0fd285daa        205bc764d14b                   "/etcd"                  14 hours ago        Up 14 hours         2379-2380/tcp, 4001/tcp, 7001/tcp              dtr-etcd-cc57c8974d24
e74563af7f91        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                                       ucp-swarm-join
1d0856dcfaf7        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp                        ucp-proxy

DDC-DTR-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                                          NAMES
c58b33ba5a29        docker/dtr-nginx:2.0.3         "/init --skip-runit /"   15 hours ago        Up 15 hours         0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp       dtr-nginx-ea5f0db3c5fb
fbdc7a59191f        docker/dtr-api:2.0.3           "/bin/server"            15 hours ago        Up 15 hours         80/tcp                                         dtr-api-ea5f0db3c5fb
cfc97b639ac7        docker/dtr-registry:2.0.3      "/init --skip-runit /"   15 hours ago        Up 15 hours         3000/tcp                                       dtr-registry-ea5f0db3c5fb
90d462424b8f        docker/dtr-rethink:2.0.3       "/start.sh"              15 hours ago        Up 15 hours         8080/tcp                                       dtr-rethinkdb-ea5f0db3c5fb
17cf28020a39        205bc764d14b                   "/etcd"                  15 hours ago        Up 15 hours         2379-2380/tcp, 4001/tcp, 7001/tcp              dtr-etcd-ea5f0db3c5fb
1c20e20be40f        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                                       ucp-swarm-join
9f0bae8a0d67        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp                        ucp-proxy
```

   b. UCP nodes have the following containers running:



```
UCP-Node-4 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
a2b3b7e5514f        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                 ucp-swarm-join
11c3222be8bf        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp  ucp-proxy

UCP-Node-2 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
4984eeabf238        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                 ucp-swarm-join
43a27493cb92        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp  ucp-proxy

UCP-Node-1 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
68b8793ce514        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                 ucp-swarm-join
81ed94d768a0        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp  ucp-proxy

UCP-Node-3 | SUCCESS | rc=0 >>
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
a4fe4f989e53        docker/ucp-swarm:1.1.4         "/swarm join --discov"   26 hours ago        Up 26 hours         2375/tcp                 ucp-swarm-join
eb2c68059228        docker/ucp-proxy:1.1.4         "/bin/run"               26 hours ago        Up 26 hours         0.0.0.0:12376->2376/tcp  ucp-proxy
```

- On UCP and DTR nodes, the infrastructure containers were restarted and the containers came up without any issues. On the controller master and replica nodes, tests were not performed to the above mentioned Docker defect.

Tests Performed on Cisco UCS Servers

Cisco UCS provides a robust system which has no single point of failure, right from Cisco Fabric Interconnects to adapters on Cisco UCS blade servers. However, there are failures which cannot be handled at the system level, such failures are:

- CPU failures
- Memory or DIMM failures
- Cisco VIC failures
- Motherboard failures

CPU/Motherboard/VIC (if only one present) failures results in blade/node getting out of service thereby causing the cluster to operate in a reduced capacity. Application containers running on them will have to be started on the other surviving nodes manually in case such a failure is encountered. A fully loaded cluster can also be on a performance tax till the required capacity is restored.

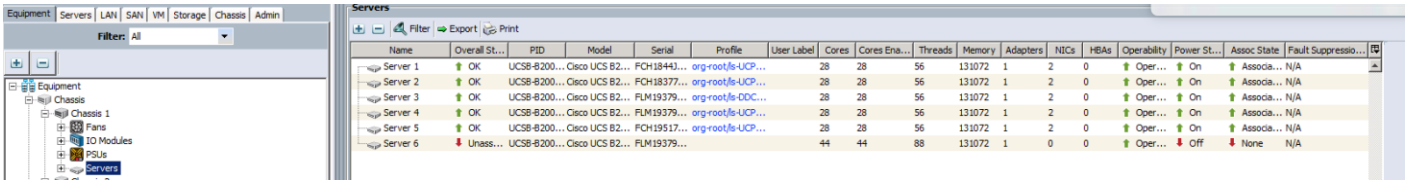
Hardware failures that are addressed at the system level include:

1. Cisco Nexus Switch – Cisco UCS Fabric interconnects are connected to upstream Cisco Nexus 9000 Series switches in a vPC mode with redundant uplinks in a portchannel configuration. It is observed that Cisco Nexus 9000 Series switch failure did not impact the application data path. Tests passed without any issues.
2. Cisco UCS Fabric Interconnect – Cisco UCS has Fabric Interconnects in redundant mode to provide no-single point of failure for the application container's data path. Since vNICs are configured for Fabric failover, in the event of any uplink, upstream Cisco Nexus 9000 Series switch and Cisco Fabric Interconnect failure data path of the application containers fails over automatically to the redundant fabric interconnect. Following tests were performed and have passed with no impact on the container application's data path:
   - While a container was pinging to the external gateway address, uplink ports on the primary Fabric Interconnect were disabled.
   - Primary Cisco Fabric Interconnect was shutdown.
   - Cisco Nexus 9000 Series switch was shutdown.
3. Cisco UCS Fabric Extenders (IOM) – Cisco UCS chassis has two IOMs providing converged connectivity to the blade and Fabric Interconnects. It carries both management control and data plane traffic to the upstream switch and acts as an extension to fabric interconnect for the blade IO. Tests were performed to test if there were any interruptions in traffic. When the containers were pinging internally to each other between the blades and to the gateway address, the IOM-A was removed and inserted back. No packet loss observed.
4. Hard Disk – All the blade nodes have RAID-1 configured for both OS boot LUN and Docker data LUN. While nodes were up and were running containers, one of the node's hard disk was removed and inserted back. Test passed with no outage at any level - OS/ DDC/ Containers.

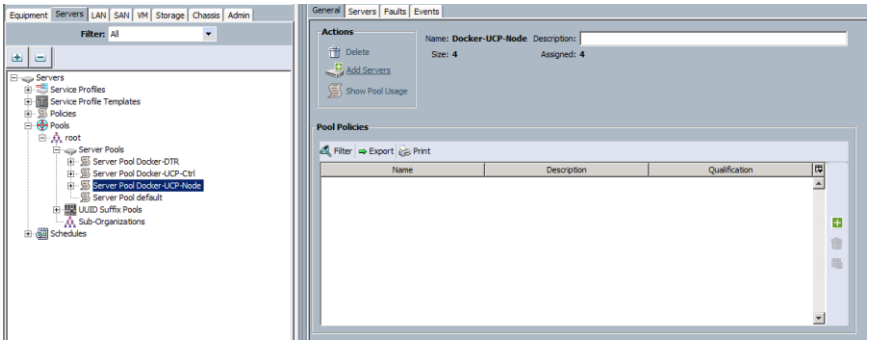
Upscale Tests

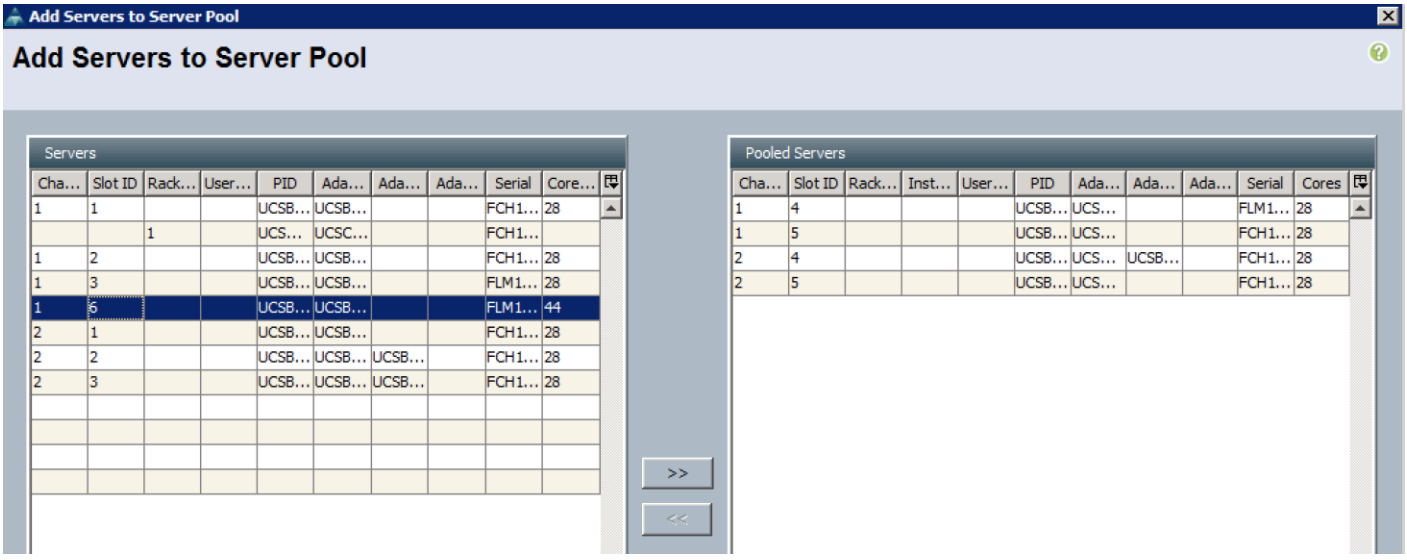
Cisco UCS infrastructure has a built-in mechanism to scale the deployed application environment through service profile templates and server pool concepts. It is very easy to scale-up the infrastructure. By inserting new blade hardware and adding it into the server pool, gets discovered automatically and gets a new service profile associated to it. Once the service profile is associated the blade gets a new node entity to be included in the Docker Datacenter pod. This procedure remains the same for any type of nodes, be it a UCP Controller, UCP or DTR node till the point where OS and Docker Engine is installed. Depending on the node type, node joining command to the existing UCP cluster varies. Blade discovery at the UCS Manager level is common; however, in order to differentiate and distribute in the two different chassis we have created three service profile templates one each for UCP Controller, UCP and DTR nodes. Service profiles are instantiated from one of these service profile templates based on the need for scale of a particular type of node. Here are the steps to scale up (for example, a UCP node):

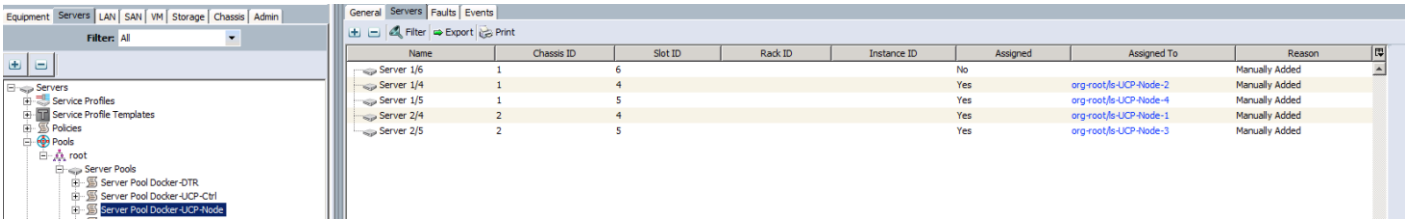
1. Insert a new blade in any available chassis slot and get it discovered:



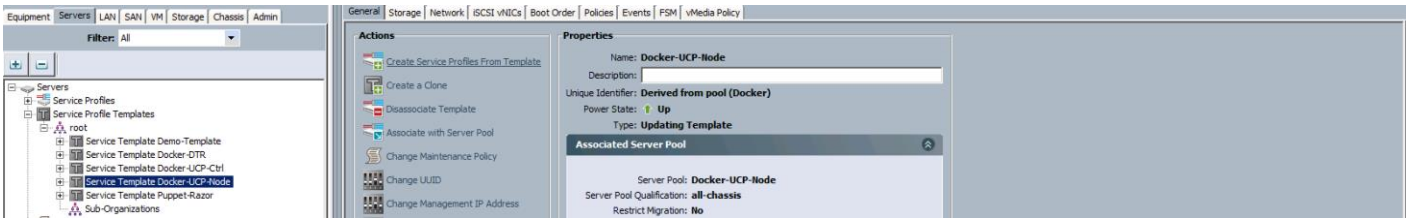
2. After successful discovery, add the server in to UCP Node server pool:

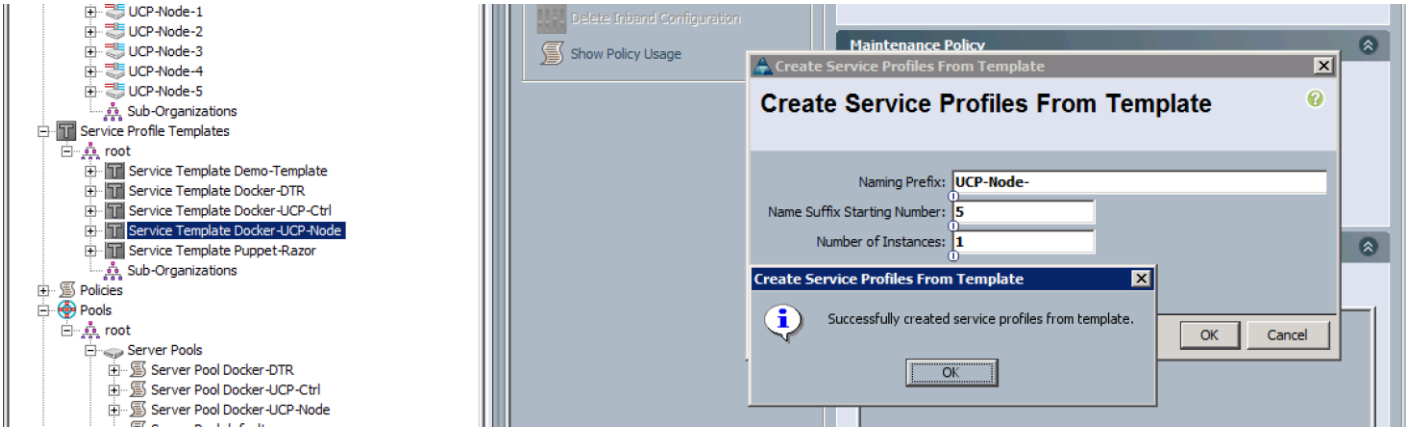




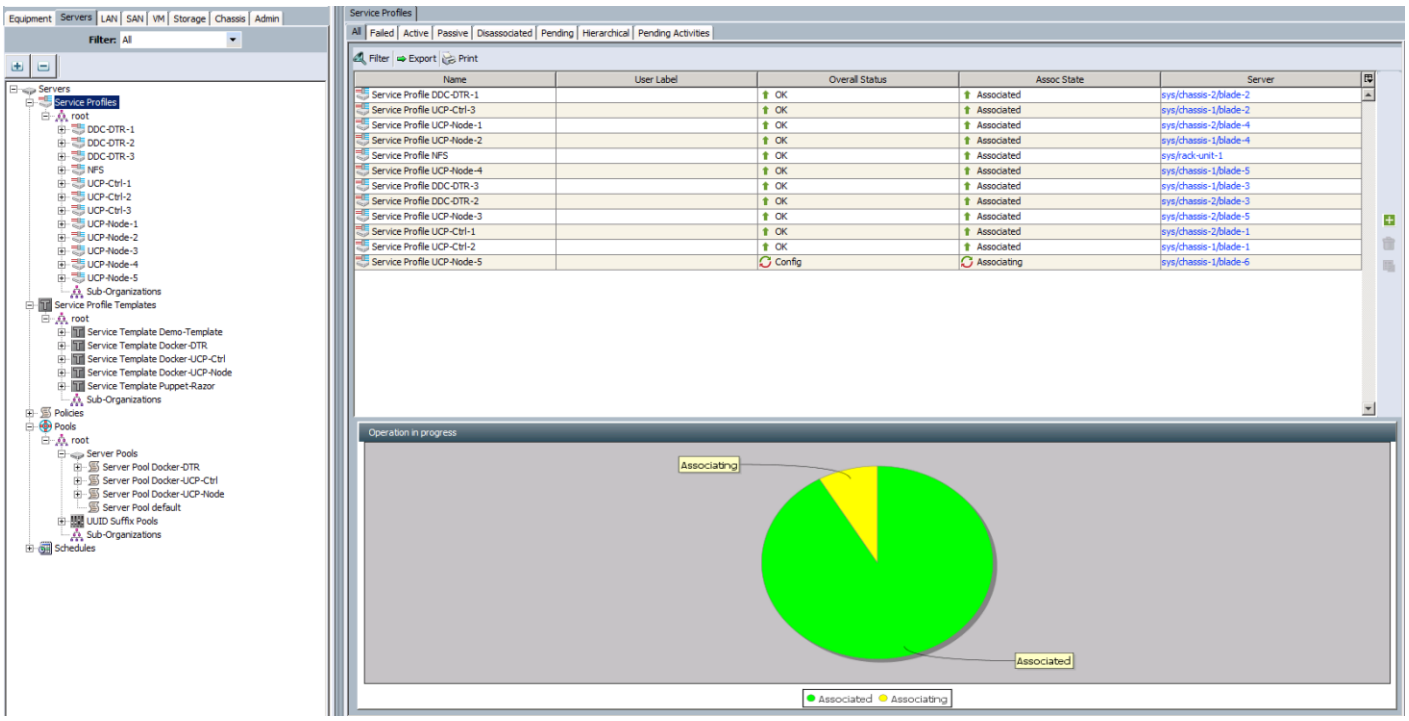


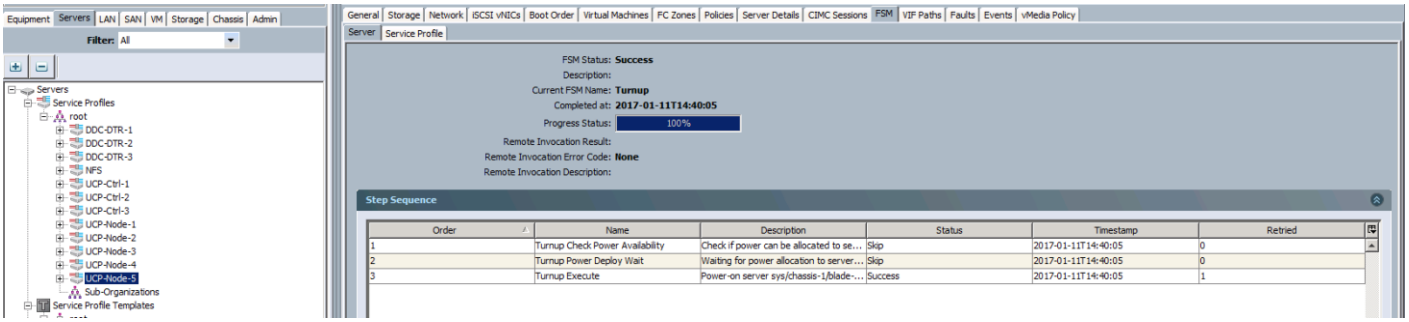
3. After successfully adding the node to the server pool, create new service profile (for example, UCP-Node-5) from the specific template based on the node type (For example, Docker-UCP-Node):

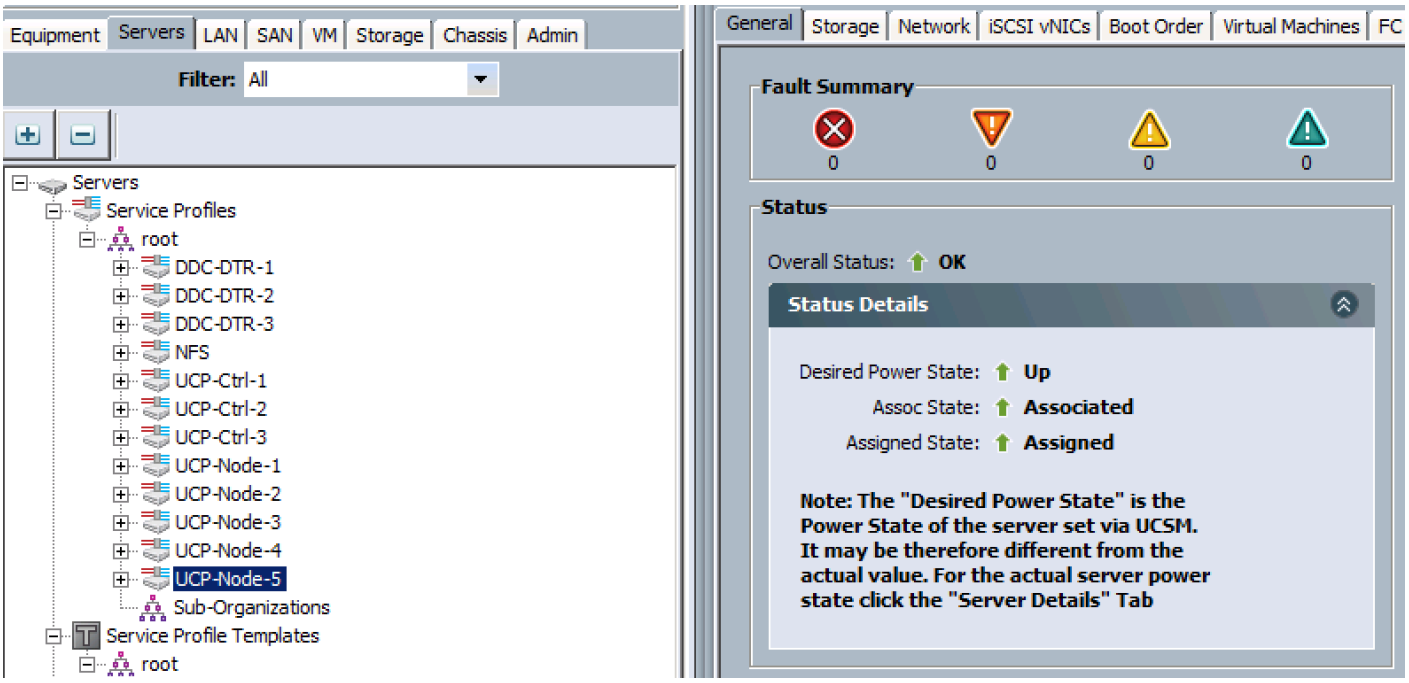




4. Service profile association will automatically kick-in as the template was already associated to the server pool.

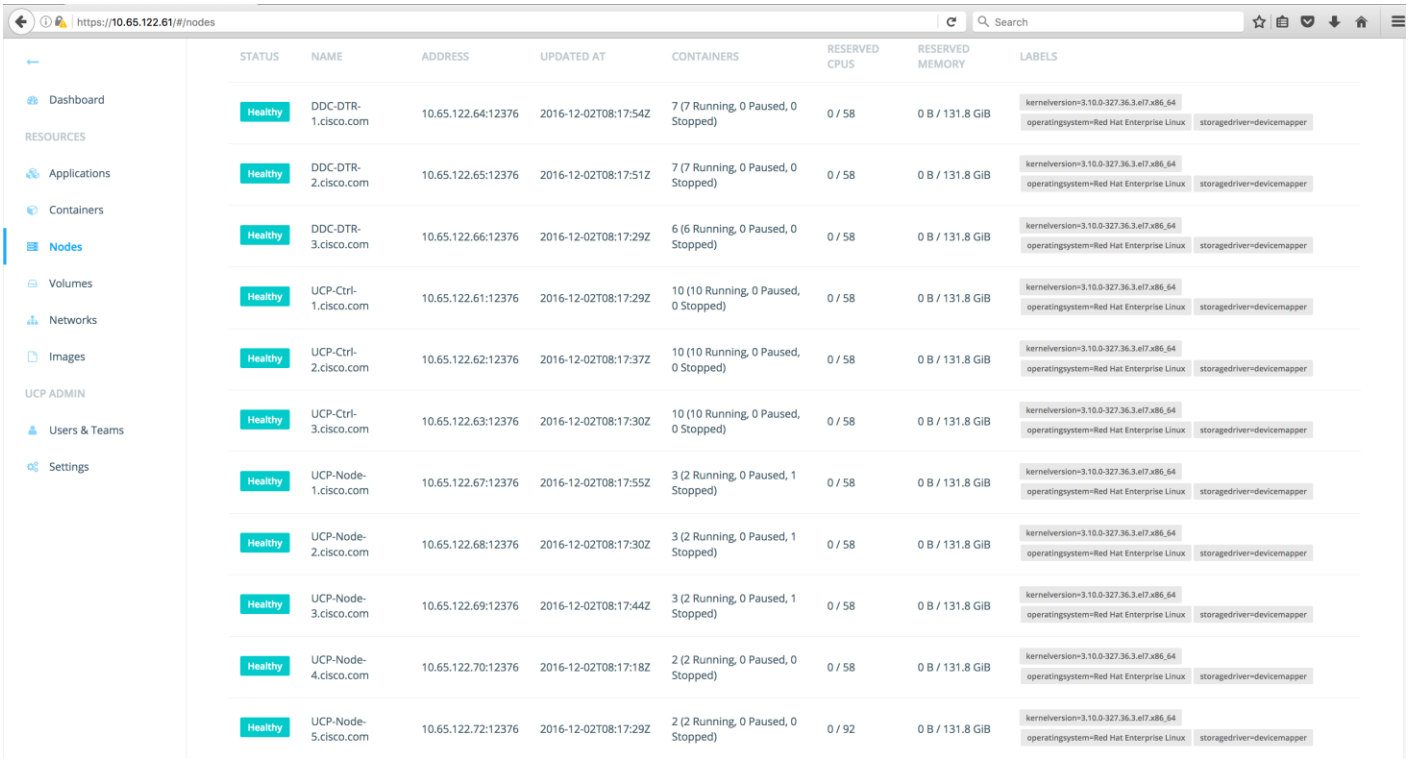

Equipment
Servers
LAN
SAN
VM
Storage
Chassis
Admin
Filter: All
Service Profiles
root
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
NFS
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
UCP-Node-5
Sub-Organizations
Service Profile Templates
Service Template Demo-Template
Service Template Docker-DTR
Service Template Docker-UCP-Ctrl
Service Template Docker-UCP-Node
Service Template Puppet-Razor
Policies
Pools
Server Pools
Server Pool Docker-DTR
Server Pool Docker-UCP-Ctrl
Server Pool Docker-UCP-Node
Server Pool default
UUID Suffix Pools
Schedules
All
Failed
Active
Passive
Disassociated
Pending
Hierarchical
Pending Activities
Filter
Export
Print
Name
User Label
Overall Status
Assoc State
Server
Service Profile DDC-DTR-1 OK Associated sys/chassis-2/blade-2
Service Profile UCP-Ctrl-3 OK Associated sys/chassis-1/blade-2
Service Profile UCP-Node-1 OK Associated sys/chassis-2/blade-4
Service Profile UCP-Node-2 OK Associated sys/chassis-1/blade-4
Service Profile NFS OK Associated sys/rack-unit-1
Service Profile UCP-Node-4 OK Associated sys/chassis-1/blade-5
Service Profile DDC-DTR-3 OK Associated sys/chassis-1/blade-3
Service Profile DDC-DTR-2 OK Associated sys/chassis-2/blade-3
Service Profile UCP-Node-3 OK Associated sys/chassis-2/blade-5
Service Profile UCP-Ctrl-1 OK Associated sys/chassis-2/blade-1
Service Profile UCP-Ctrl-2 OK Associated sys/chassis-1/blade-1
Service Profile UCP-Node-5 Config Associating sys/chassis-1/blade-6
Operation in progress
Associating
Associated
Associated
Associating


Equipment
Servers
LAN
SAN
VM
Storage
Chassis
Admin
Filter: All
Servers
Service Profiles
root
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
NFS
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
UCP-Node-5
Sub-Organizations
Service Profile Templates
General
Storage
Network
iSCSI vNICs
Boot Order
Virtual Machines
FC Zones
Policies
Server Details
CIMC Sessions
FSM
VIF Paths
Faults
Events
vMedia Policy
Server
Service Profile
FSM Status: Success
Description:
Current FSM Name: Turnup
Completed at: 2017-01-11T14:40:05
Progress Status: 100%
Remote Invocation Result:
Remote Invocation Error Code: None
Remote Invocation Description:
Step Sequence
Order
Name
Description
Status
Timestamp
Retried
1 Turnup Check Power Availability Check if power can be allocated to se... Skip 2017-01-11T14:40:05 0
2 Turnup Power Deploy Wait Waiting for power allocation to server... Skip 2017-01-11T14:40:05 0
3 Turnup Execute Power-on server sys/chassis-1/blade-... Success 2017-01-11T14:40:05 1


Equipment
Servers
LAN
SAN
VM
Storage
Chassis
Admin
Filter: All
Servers
Service Profiles
root
DDC-DTR-1
DDC-DTR-2
DDC-DTR-3
NFS
UCP-Ctrl-1
UCP-Ctrl-2
UCP-Ctrl-3
UCP-Node-1
UCP-Node-2
UCP-Node-3
UCP-Node-4
UCP-Node-5
Sub-Organizations
Service Profile Templates
root
General
Storage
Network
iSCSI vNICs
Boot Order
Virtual Machines
FC
Fault Summary
0
0
0
0
Status
Overall Status: OK
Status Details
Desired Power State: Up
Assoc State: Associated
Assigned State: Assigned
Note: The "Desired Power State" is the Power State of the server set via UCSM. It may be therefore different from the actual value. For the actual server power state click the "Server Details" Tab

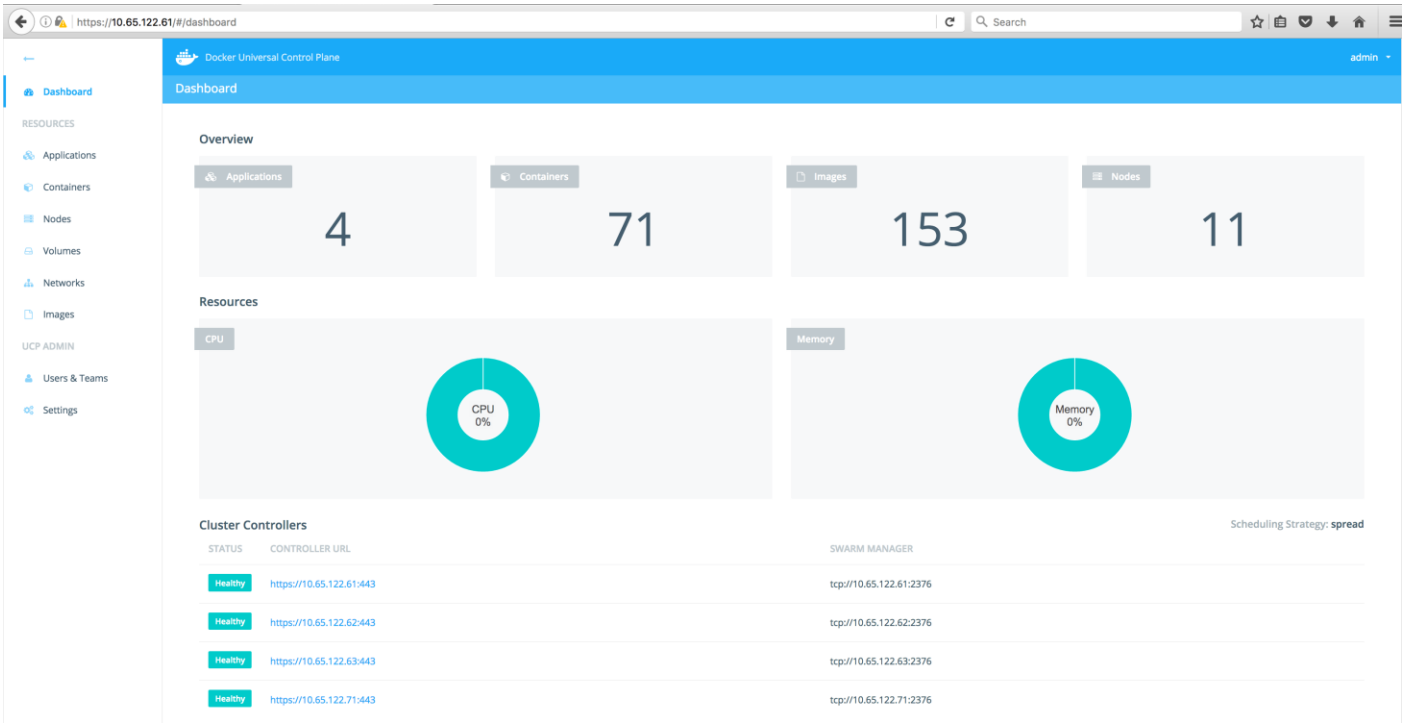
5. Follow the OS install procedure explained in the earlier section to install the RHEL7.2 stock operating system on the blade.
6. Once the Operating System gets installed, perform post-install configuration tasks, such as subscribing to RHN, updating the system, configuring NTP, updating /etc/hosts file to include all the cluster nodes, open up firewall ports etc as detailed in the earlier section.
7. Install Docker engine repo, as explained in the earlier section.
8. Install Docker Engine and configure Docker data LUN of sixe 200 GB size, using device mapper storage driver in 'direct-lvm' mode as described in the earlier section.
9. Enable and start Docker-Engine and follow the remaining steps to include the new node into the cluster:

```
[root@UCP-Node-5 ~]# docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:1.1.4 join -i
Unable to find image 'docker/ucp:1.1.4' locally
1.1.4: Pulling from docker/ucp

c0cb142e4345: Pull complete
bcee58a6e550: Pull complete
dbd82e351077: Pull complete
Digest: sha256:d16275fabledac53cd19e7b408581fb3a115013c721fabbc1e4f813abe272065
Status: Downloaded newer image for docker/ucp:1.1.4
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
WARN[0000] Your system uses devicemapper.  We can not accurately detect available storage space.  Please make sure you have at least 3.00 GB
available in /var/lib/docker
Please enter the URL to your UCP server: https://10.65.122.61:443
UCP server https://10.65.122.61:443
CA Subject: UCP Client Root CA
Serial Number: 50e994a3aaf9b501
SHA-256 Fingerprint=F9:38:76:95:E2:17:80:F7:3F:43:2F:E9:EF:E4:1C:15:13:D6:E5:63:59:64:33:5A:8C:EB:84:D2:E3:4C:F5:31
Do you want to trust this server and proceed with the join? (y/n): y
Please enter your UCP Admin username: admin
Please enter your UCP Admin password:
INFO[0031] Pulling required images... (this may take a while)
We detected the following hostnames/IP addresses for this system [UCP-Node-5.cisco.com 127.0.0.1 172.17.0.1 10.65.122.72]

You may enter additional aliases (SANs) now or press enter to proceed with the above list.
Additional aliases:
INFO[0001] This engine will join UCP and advertise itself with host address 10.65.122.72 - If this is incorrect, please specify an alternativ
e address with the '--host-address' flag
INFO[0001] Verifying your system is compatible with UCP
INFO[0006] Starting local swarm containers
INFO[0008] New configuration established.  Signalling the daemon to load it...
INFO[0009] Successfully delivered signal to daemon
[root@UCP-Node-5 ~]#
```



10. For adding another controller node, all the above steps remain same except that while adding a new node to the cluster we need to run the following command as shown in the screenshot below:

```
[root@UCP-Ctrl-4 ~]# docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock -v /root/backup.tar:/backup.tar docker/ucp:
1.1.4 join --interactive --replica --passphrase "secret" --fresh-install
INFO[0000] Your engine version 1.12.3-cs4, build 65c6c4c (3.10.0-327.36.3.el7.x86_64) is compatible
WARN[0000] Your system uses devicemapper.  We can not accurately detect available storage space.  Please make sure you have at least 3.00 GB
 available in /var/lib/docker
Please enter the URL to your UCP server: https://10.65.122.61:443
UCP server https://10.65.122.61:443
CA Subject: UCP Client Root CA
Serial Number: 50e994a3aaf9b501
SHA-256 Fingerprint=F9:38:76:95:E2:17:80:F7:3F:43:2F:E9:EF:E4:1C:15:13:D6:E5:63:59:64:33:5A:8C:EB:84:D2:E3:4C:F5:31
Do you want to trust this server and proceed with the join? (y/n): y
Please enter your UCP Admin username: admin
Please enter your UCP Admin password:
INFO[0029] All required images are present
We detected the following hostnames/IP addresses for this system [UCP-Ctrl-4.cisco.com 127.0.0.1 172.17.0.1 10.65.122.71]

You may enter additional aliases (SANs) now or press enter to proceed with the above list.
Additional aliases:
INFO[0001] This engine will join UCP and advertise itself with host address 10.65.122.71 - If this is incorrect, please specify an alternati
ve address with the '--host-address' flag
INFO[0001] Verifying your system is compatible with UCP
INFO[0001] Checking that required ports are available and accessible
INFO[0038] Starting local swarm containers
INFO[0041] Starting UCP Controller replica containers
WARN[0099] Configuration updated. You will have to manually restart the docker daemon for the changes to take effect.
WARN[0099] If you have to restart daemons on multiple controllers, restart them one by one to prevent long restart times.
[root@UCP-Ctrl-4 ~]#
```



Removal of Docker UCP Controller Replicas and UCP Nodes

UCP Node Removal – This does not require any special steps. UCP 'uninstall' command is needed to be run on the UCP node which is getting removed from the cluster. Cluster management control plane automatically removes the said node from monitoring and scheduling purposes after fixed timeout period. Inventory also gets refreshed accordingly.

```
docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock
docker/ucp:1.1.4 uninstall -i
```

UCP Controller Replica Removal – There are 2 stages to do this. In case of cluster scale down scenarios, where user wants to remove a replica controller node, first ucp uninstallation needed to be run on the node, after that removed replica node records are required to be cleaned up from the existing cluster etcd(key-pair store database). In case of removal of failed replica node scenarios, only database cleanup is needed

Uninstalling UCP on replica node:

```
docker run --rm -it --name ucp -v /var/run/docker.sock:/var/run/docker.sock
docker/ucp:1.1.4 uninstall -i
```

Cleaning up etcd/kv state store data base:

1. On one of the controller nodes check the overall cluster status:

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem ls /orca/v1/controllers
```

To see the cluster health:

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem cluster-health
```

Identify the failed node entry by running the above command.

2. Remove the node using the command:

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem rm /orca/v1/controllers/<node-ip>
```

The above command removes the desired controller replica node from the user interface.

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem rm /orca/v1/config/clientca_<node-ip>:12381

docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem rm /orca/v1/config/clientca_<node-ip>:12382
```

The above commands remove the references of the failed nodes permanently from the state store database.

3. Remove the failed node from the cluster PKI member list:

   To list the members:

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem member list
```

   Removal of the failed member:

```
docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem member list

ETCDCTL="docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --
ca-file /etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem"

echo $ETCDCTL

CONTROLLER_IP="<node-ip>"

${ETCDCTL} member remove <node-id as shown in the cluster health status
command for the failed node>

${ETCDCTL} rm /orca/v1/config/clientca_$CONTROLLER_IP:12381

${ETCDCTL} rm /orca/v1/config/clientca_$CONTROLLER_IP:12382

${ETCDCTL} rm /orca/v1/controllers/$CONTROLLER_IP

${ETCDCTL} rm /orca/v1/swarm/0/managers/$CONTROLLER_IP:2376

docker exec -it ucp-kv etcdctl --endpoint https://127.0.0.1:2379 --ca-file
/etc/docker/ssl/ca.pem --cert-file /etc/docker/ssl/cert.pem --key-file
/etc/docker/ssl/key.pem member list
```

The above command confirms that the cluster PKI member list is clean.

Bill of Materials

The following infrastructure components are needed for UCS B-Series first architecture:

Component	Model	Quantity	Comments
Docker Datacenter UCP Controller Nodes	B200M4 (UCSB-B200-M4)	3	CPU – 2 x E5-2660 V4 (UCS-CPU-E52660E) Memory – 8 x 16GB 2133 DIMM – total of 128G (UCS-MR-1X162RU-A) Local Disks – 2 x 300 GB SAS disks for Boot (UCS-HDD300GI2F105) Network Card – 1x1340 VIC (UCSB-MLOM-40G-03) Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)
Docker Datacenter DTR Nodes	B200M4 (UCSB-B200-M4)	3	CPU – 2 x E5-2660 V4 (UCS-CPU-E52660E) Memory – 8 x 16GB 2133 DIMM – total of 128G (UCS-MR-1X162RU-A) Local Disks – 2 x 300 GB SAS disks for Boot (UCS-HDD300GI2F105) Network Card – 1x1340 VIC (UCSB-MLOM-40G-03) Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)
Docker Datacenter UCP Nodes	B200M4 (UCSB-B200-M4)	4	CPU – 2 x E5-2660 V4 (UCS-CPU-E52660E) Memory – 8 x 16GB 2133 DIMM – total of 128G (UCS-MR-1X162RU-A) Local Disks – 2 x 300 GB SAS disks for Boot (UCS-HDD300GI2F105) Network Card – 1x1340 VIC (UCSB-MLOM-40G-03) Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)
Chassis	UCS 5108 (N20-C6508)	2	
IO Modules	IOM 2208XP (UCS-IOM-2208XP)	4	
Fabric Interconnects	UCS 6248UP (UCS-FI-6248UP)	2	

Switches	Nexus 9372PX (N9K-C9396PX)	2	
Docker Server/On-Prem Subscription	Docker Datacenter Subscription	1	Includes Docker UCP, DTR, CS Engine https://store.docker.com/bundles/docker-datacenter

The following infrastructure components are needed for UCS C-Series second architecture:

Component	Model	Quantity	Comments
Docker Datacenter UCP Controller + DTR Nodes	C220M4 (UCSC-C220-M4S)	3	CPU – 2 x E5-2669 V4 (UCS-CPU-E52699E) Memory – 16 x 16GB 2133 DIMM – total of 256GB (UCS-MR-1X162RU-A) Local Disks – 6 x 1.2 TB SAS Disks (UCS-HD12TB10K12G) Network Card – 1x1227 VIC (UCSC-MLOM-CSC-02) Raid Controller – Cisco MRAID 12G SAS Controller (UCSC-MRAID12G)
Docker Datacenter UCP Nodes	C220M4 (UCSC-C220-M4S)	1	CPU – 2 x E5-2669 V4 (UCS-CPU-E52699E) Memory – 16 x 16GB 2133 DIMM – total of 256GB (UCS-MR-1X162RU-A) Local Disks – 6 x 1.2 TB SAS Disks (UCS-HD12TB10K12G) Network Card – 1x1227 VIC (UCSC-MLOM-CSC-02) Raid Controller – Cisco MRAID 12G SAS Controller (UCSC-MRAID12G)
Fabric Interconnects	UCS 6248UP (UCS-FI-6248UP)	2	
Switches	Nexus 9372PX (N9K-C9396PX)	2	
Docker Server/On-Prem Subscription	Docker Datacenter Subscription	1	Includes Docker UCP, DTR, CS Engine https://store.docker.com/bundles/docker-

			datacenter

Summary

The emergence of the Docker platform and the underlying support in the Linux and Windows kernel has enabled a shift in the way that traditional applications are managed and new applications are designed and built, moving to more efficient Microservices architectures. Microservices architectures are an approach to modernizing and building complex applications through small, independent components that communicate with each other over language-independent APIs.

The integration of Docker Datacenter with Cisco UCS converged infrastructure enabled through automation tools like UCS Python SDK and Ansible provides container application platform to get deployed and managed quickly at scale. It also provides a very good starting point for enterprise IT to make in-roads into DevOps and CI/CD model for application environment for quick business need turn around.

Cisco and Docker's container management solution, Docker Datacenter, accelerates your IT transformation by enabling easier and faster deployments, greater flexibility, heightened business agility, increased efficiency with lowered risk and higher ROI for enterprise customers.

About Authors

Rajesh Kharya, Technical Leader, Cisco UCS Solutions Engineering, Cisco Systems Inc.

Seasoned data center Infrastructure professional specializing in Unix/Linux, Networking, Unix File systems, Storage Management and Cisco's Converged compute/storage/network/virtualization platform – Cisco UCS. Functional roles include Unix/Linux administration, Software Feature QA leadership to Solutions Development and Architecture. Currently involved in Solution development projects Containers on UCS baremetal and OpenStack on UCS platform. With over 15 years of experience in the domain, excited and glad to be part of technological transitions happening in the datacenter.

Uday Shetty, Senior Software Engineer, Strategic Development, Docker, Inc.

Uday Shetty has more than 25+ years of experience in Enterprise Applications at Sun/Oracle and Docker, working with ISVs and Strategic Partners on new technology adoptions and Reference Architectures. Uday's expertise includes Java, J2EE, Docker for Linux/Windows, Docker Datacenter for Enterprises and building Docker Datacenter templates for Cloud providers.

Sindhu Sudhir, Technical Writer/TME, Cisco Systems Inc.

Sindhu has 7+ years of experience in the IT industry including technical documentation/ editing, software development, infrastructure automation and has strong knowledge of networking. Her recent contributions to the Cisco UCS solution engineering team include co-authoring UCS C240 I/O characterization and Containers on UCS platform whitepapers.

Acknowledgements

- Cisco Systems: Vadi Bhatt, Vishwanath Jakka
- Docker: Bradley Wong