

Cisco UCS Infrastructure with Contiv and Docker Enterprise Edition for Container Management

Design and Deployment Guide for Cisco UCS Infrastructure with Contiv Container Networking and Docker Enterprise Edition 17.06

Last Updated: May 1, 2018



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>.

ALL DESIGNS, SPECIFICATIONS, STATEMENTS, INFORMATION, AND RECOMMENDATIONS (COLLECTIVELY, "DESIGNS") IN THIS MANUAL ARE PRESENTED "AS IS," WITH ALL FAULTS. CISCO AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE. IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THE DESIGNS, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE DESIGNS ARE SUBJECT TO CHANGE WITHOUT NOTICE. USERS ARE SOLELY RESPONSIBLE FOR THEIR APPLICATION OF THE DESIGNS. THE DESIGNS DO NOT CONSTITUTE THE TECHNICAL OR OTHER PROFESSIONAL ADVICE OF CISCO, ITS SUPPLIERS OR PARTNERS. USERS SHOULD CONSULT THEIR OWN TECHNICAL ADVISORS BEFORE IMPLEMENTING THE DESIGNS. RESULTS MAY VARY DEPENDING ON FACTORS NOT TESTED BY CISCO.

CCDE, CCENT, Cisco Eos, Cisco Lumin, Cisco Nexus, Cisco StadiumVision, Cisco TelePresence, Cisco WebEx, the Cisco logo, DCE, and Welcome to the Human Network are trademarks; Changing the Way We Work, Live, Play, and Learn and Cisco Store are service marks; and Access Registrar, Aironet, AsyncOS, Bringing the Meeting To You, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unified Computing System (Cisco UCS), Cisco UCS B-Series Blade Servers, Cisco UCS C-Series Rack Servers, Cisco UCS S-Series Storage Servers, Cisco UCS Manager, Cisco UCS Management Software, Cisco Unified Fabric, Cisco Application Centric Infrastructure, Cisco Nexus 9000 Series, Cisco Nexus 7000 Series, Cisco Prime Data Center Network Manager, Cisco NX-OS Software, Cisco MDS Series, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, iQuick Study, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy, Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert, StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0809R)

© 2017 Cisco Systems, Inc. All rights reserved.

Table of Contents

Executive Summary	7
Business Challenges	7
Our Solution.....	8
Implementation Overview	9
Solution Benefits.....	9
Audience	10
Purpose of this Document.....	10
Solution Overview.....	11
Introduction	11
What's New?	12
Solution Components	12
Technology Overview	14
Cisco Unified Computing System.....	14
Cisco UCS Manager	15
Cisco UCS Fabric Interconnects	15
Cisco UCS 5108 Blade Server Chassis	16
Cisco UCS B200 M5 Blade Server	16
Cisco UCS C220 M5 Rack-Mount Server	17
Cisco UCS Fabric Extenders	17
Cisco VIC Interface Cards.....	18
Cisco UCS Differentiators	18
Cisco Nexus 9000 Switches	20
Cisco Contiv	20
Docker Enterprise Edition.....	22
Docker EE Engine	24
Docker Universal Control Plane (UCP)	26
Docker Trusted Registry (DTR)	31
Ansible	35
Solution Design.....	38
Architectural Overview.....	38
Cisco UCS and Docker Enterprise Edition	38
Cisco Contiv	39
Physical Topology.....	40
Logical Topology	43
Ansible Playbook	46
Ansible Playbook Tree Structure.....	47
Ansible Playbook Global Variables (group_vars)	48

Ansible Playbook Roles	51
Sizing Considerations	52
Software and Hardware Versions.....	53
Solution Deployment.....	57
Cisco Nexus 9372PX	57
Initial Configuration and Setup	57
Feature Enablement.....	58
VLAN Creation	58
Configure VPC.....	58
Configure Network Interfaces	60
Cisco UCS Manager - Administration	63
Initial Setup of Cisco Fabric Interconnects	63
Configure Ports for Server, Network and Storage Access.....	63
Cisco UCS Manager – Synchronize to NTP	64
Upgrading Cisco UCS Manager	64
Assigning Block of IP addresses for KVM Access	64
Editing Chassis Discovery Policy.....	65
Acknowledging Cisco UCS Chassis	65
Enabling Server Ports	65
Enabling Uplink Ports to Cisco Nexus 9000 Series Switches.....	66
Configuring Port Channels on Uplink Ports to Cisco Nexus 9000 Series Switches	67
Cisco UCS Configuration – LAN	69
Creating VLANs	69
Creating LAN Pools	71
Creating LAN Policies	72
Creating vNIC Templates.....	72
Cisco UCS Configuration – Server.....	74
Creating Server Policies.....	74
Creating BIOS Policy.....	74
Creating Boot Policy	75
Creating Host Firmware Package Policy	76
Creating UUID Suffix Pool	77
Creating Server Pools.....	78
Cisco UCS Configuration – Storage.....	80
Creating Storage Profile	80
Creating Service Profile Templates	87
Creating Service Profile Template for UCP Manager/Master Nodes.....	87
Creating Service Profile Template for DTR Nodes.....	100
Creating Service Profile Template for UCP Worker Nodes	101

Configuring PXE-less Automated OS Installation Infra with UCSM vMedia Policy	102
Prerequisites	102
Web Server – Installation and Configuration	102
Create Images	103
Service Profile Instantiation and Association	108
Service Profile Instantiation.....	108
Installation of Red Hat Enterprise Linux Operating System	111
Docker Enterprise Edition Installation	115
Configuring Firewall Ports for Docker EE.....	115
Ansible Installation.....	117
Ansible Playbook Execution	119
Verifying Docker Enterprise Edition Installation.....	123
Contiv Installation	127
Validation.....	132
Application Container Deployment Using Contiv	132
Contiv Network Back-end without Contiv Policy Rules.....	132
Contiv Network Back-end with Contiv Policy Rules.....	139
Test Plan	141
Functional Test Scenarios – Docker EE/ Contiv	142
Scale Tests.....	142
High-Availability Tests.....	146
Bill of Materials	150
Addendum.....	153
UCS with Contiv and DEE - Hybrid cluster with Baremetal and VMs.....	153
Mixed Cluster Solution Components	153
Solution Design	153
Physical Topology.....	153
Technical Requirements for Virtual Environment	155
Logical topology	157
Solution Deployment.....	158
UCS Manager Configuration	158
VMware ESXi Installation and Virtual Machine Deployment	163
Docker EE Installation on Virtual Machines and Adding them to Existing UCP Cluster	173
Contiv Installation on New Nodes and Adding them to Existing Contiv Cluster	174
Testing and Validation for mixed cluster environment	175
Bill of Material - Additional Components for Bare Metal and Virtual Machine Cluster	177
Summary	179
Appendix	180
Appendix - I: HAProxy Example Configuration for External Load-Balancer.....	180

Appendix - 2: Ansible Playbook Host Inventory and Task Execution YAML File	183
Appendix - 3: Contiv Data Path Troubleshooting	185
About the Authors.....	192
Acknowledgements	192

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

Docker Enterprise Edition (EE) 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 the Docker EE platform 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 EE 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 Enterprise Edition provides native container management tools, including the Docker Trusted Registry (DTR) and Universal Control Plane (UCP) which included Docker Swarm as an orchestration component. 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).

Contiv unifies containers, bare metal and virtual machine environments with a single networking fabric. This allows container network to be addressable from existing bare metal and/or virtual machine networks. Cisco UCS servers with Contiv network plugin for Docker containers provide rich policy framework for container networking for the enterprise grade application environment. Contiv is an open source Docker certified network plugin and is available through Apache 2 license. Contiv can be sourced either from Docker store or Contiv github.

Cisco and Docker have joined hands to offer Container Management Solution on Cisco UCS Infrastructure with Docker Enterprise Edition. This enables enterprises to modernize traditional applications and build microservices architecture using the Docker EE platform **and tools on Cisco's proven** Cisco UCS Integrated Infrastructure. The combination of Docker EE and Cisco UCS server hardware enables a highly scalable, resilient, and elastic application deployment environment with the simplicity of the on-premises 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:

- **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.
- **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-premises 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.

Our Solution

This Cisco Validated Design focuses on Cisco UCS Integrated Infrastructure with Contiv Container Networking for Docker Enterprise Edition that enables enterprises to rapidly deploy and manage production ready Container as a Service environment on a highly available and secure platform. Contiv provides a higher-level of networking abstraction for containerized applications and microservices. It secures applications using rich policy framework with built-in service discovery and service routing features for scale out services. Additionally, it helps automate DevOps process with an integrated security from development to production, accelerating product release cycle 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 and Docker collaborated to deploy Docker Enterprise Edition (which includes Docker Universal Control Plane, Docker Trusted Registry and Docker Engine on Cisco UCS platform).

This solution covers production ready installation, provisioning, configuring and deploying application containers using Docker Enterprise Edition container platform on Cisco UCS B-Series and C-Series servers in two separate topologies for production and dev/test use cases. Contiv is being used as plugin for **providing native networking for the containers and microservices. Docker Enterprise Edition'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:

- Automated, quick and easy installation of Cisco UCS Integrated Infrastructure, Docker Enterprise Edition and Contiv Network Plugin and Application Containers

- Application Container Management through Docker Enterprise Edition on compute nodes irrespective of form factors by utilizing Cisco UCS Manager capabilities
- Create and configuring tenants, networks, isolation policies and storage across complete infrastructure for Application Containers
- High-Availability test inducing nodes, network devices, Contiv components 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, Contiv and Docker Enterprise Edition 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

Cisco UCS Integrated Infrastructure with Contiv for Docker Enterprise Edition solution is an integrated and validated design and deployment guide for the enterprise to run application containers and microservices at production grade. This solution is implemented on Cisco UCS B- and C-Series servers and Cisco Nexus switch platforms with Contiv network plugin. 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 templates, Policies for network access and Storage access to run bare metal OSes. Operating System installation, post OS install configuration, Docker Enterprise Edition installation and Contiv network plugin installation, all such tasks are automated through Ansible automation framework and playbooks. The end-to-end stack is tested for compatibility and interoperability (recommended software stack), performance, and scalability, high-availability and security policies for application containers for network access through Contiv network plugin. The containers are deployed and managed through Docker tools like Docker Compose, UCP CLI/UI etc. The deployment guide provides step by step instructions on setting up the complete stack and solution validation test results.

Solution Benefits

The benefits of Cisco UCS and Contiv with Docker Enterprise Edition include the following:

- Contiv
 - Rich Policy Framework: Set bandwidth and isolation policies in a multi-tenant environment
 - Multi-Infrastructure: Bare Metal, Virtual Machines and Containers
 - Multi-Network Support: Layer 2, Layer 3, BGP and ACI
 - Multi-Platforms: Docker, Kubernetes, OpenShift and more
 - Open Source: Available under Apache 2 License on GitHub
- Cisco UCS
 - Simplicity: Reduced datacenter complexities through Cisco UCS 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 Enterprise Edition
 - 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 this Document

This document highlights the benefits of using Cisco UCS infrastructure with Contiv for Docker Enterprise Edition to efficiently deploy, scale, and manage a production-ready application container environment for enterprise customers. While Cisco UCS infrastructure provides a platform for compute, network and storage needs, Contiv delivers policy based networking for the containers. The goal of this document is to demonstrate the value that Cisco UCS brings to the data center, such as single-point hardware lifecycle management and highly available compute and network infrastructure for application container deployments using Docker Enterprise Edition and Contiv.

Solution Overview

Introduction

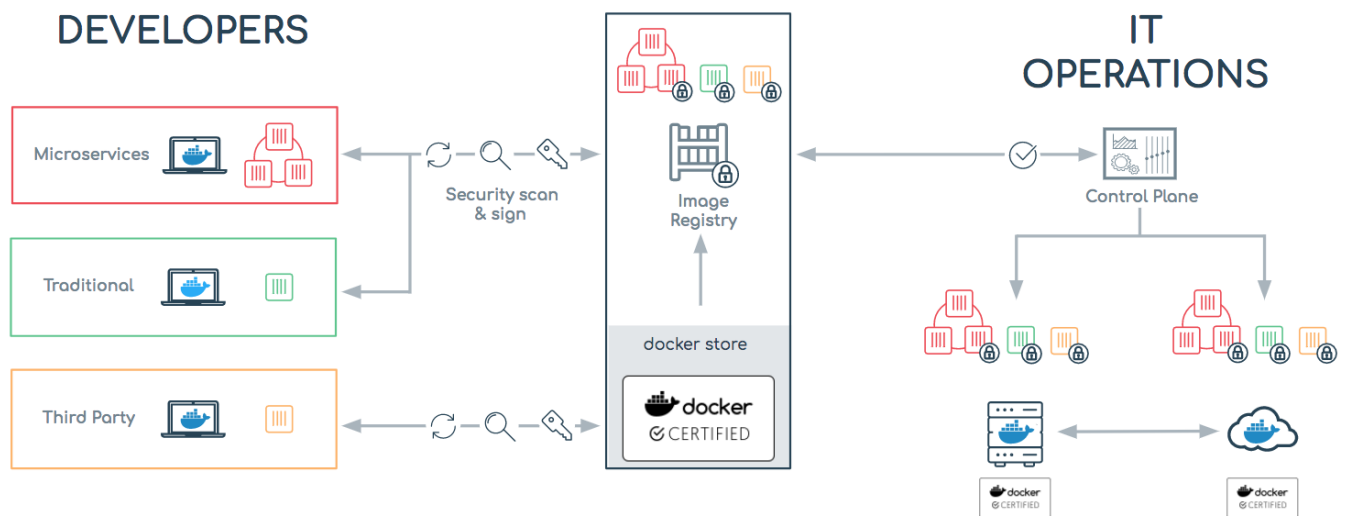
Cisco UCS Integrated Infrastructure solutions speed up IT operations today and create the modern technology foundation you need for initiatives like private cloud, big data, and desktop virtualization. With Cisco UCS Manager and Cisco Single Connect Technology, hardware is automatically configured by 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 bare metal, 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 containerized applications on-premise or in cloud environment.

Figure 1 Build Secure Software Supply Chain with Docker



Docker Enterprise Edition 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 Enterprise Edition integrates to enterprise business – from on-premises and VPC deployment models, open APIs and interfaces, to flexibility for supporting a wide variety of workflows.

With the advent of containers and microservices architecture, there is a need of automated or programmable network infrastructure specifically catering to dynamic workloads which can be formed using containers. With container and microservices technologies, speed and scale becomes critical. Because of these requirements, Automation becomes a critical component in the Network provisioning for future workloads. Contiv unifies containers, VMs, and bare metal with a single networking fabric, allowing container networks to be addressable from VM and bare metal networks. Contiv combines strong network performance, support for industry-leading hardware, and an application-oriented policy that can move across networks with the application. Multi-tenancy and separate data path for application container/microservices I/O are the key feature of Contiv.

Automation need for deploying and configuring infrastructure for Docker Enterprise Edition and Contiv has been addressed in this solution via built-in UCSM features for operating system deployment. And post installation configuration tasks and DEE stack including Contiv is being provisioned through open source Ansible automation engine. Users are free to create their own Ansible playbook to achieve the same results or can download the entire playbook for readymade use cases from Cisco UCS GitHub site.

What's New?

- Automated bare metal operating system provisioning
- Ansible Play books and installer for Storage configuration, Docker Enterprise Edition stack and Contiv deployments
- Automated scale-up/down infrastructure and Docker & Contiv stacks

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 6332-16UP Fabric Interconnects
- Cisco 2304XP IO Module or Cisco UCS Fabric Extenders
- Cisco B200 M5 Servers
- Cisco C220 M5S Servers
- Cisco VIC 1340
- Cisco VIC 1385

- Cisco Nexus 9396PX Switches
- Docker Enterprise Edition (DEE)
- Docker EE Engine
- Docker Universal Control Plane (UCP)
- Docker Trusted Repository (DTR)
- Red Hat Enterprise Linux 7.3
- Contiv v2plugin 1.1.7

Technology Overview

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

Cisco Unified Computing System

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

- Comprehensive Management
- Radical Simplification
- High Performance

The Cisco Unified Computing System consists of the following components:

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

- Management - the system uniquely integrates all system components to enable the entire solution to be managed as a single entity by the Cisco UCS Manager. The Cisco UCS Manager has an intuitive graphical user interface (GUI), a command-line interface (CLI), and a powerful scripting library module for Microsoft PowerShell built on a robust application programming interface (API) to manage all system configuration and operations.

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

Cisco UCS Manager

Cisco Unified Computing System (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 6300 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><http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-manager/index.html>

Cisco UCS Fabric Interconnects

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

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

For more information, visit the following link: <https://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-6332-16up-fabric-interconnect/index.html>

Cisco UCS 5108 Blade Server Chassis

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

For more information, 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 B200 M5 Blade Server

The Cisco UCS B200 M5 Blade Server delivers performance, flexibility, and optimization for deployments in data centers, in the cloud, and at remote sites. This enterprise-class server offers market-leading performance, versatility, and density without compromise for workloads including Virtual Desktop Infrastructure (VDI), web infrastructure, distributed databases, converged infrastructure, and enterprise applications such as Oracle and SAP HANA. The B200 M5 server can quickly deploy stateless physical and virtual workloads through programmable, easy-to-use Cisco UCS Manager Software and simplified server access through Cisco SingleConnect technology. The Cisco UCS B200 M5 server is a half-width blade. Up to eight servers can reside in the 6-Rack-Unit (6RU) Cisco UCS 5108 Blade Server Chassis, offering one of the highest densities of servers per rack unit of blade chassis in the industry. You can configure the B200 M5 to meet your local storage requirements without having to buy, power, and cool components that you do not need. The B200 M5 provides you these main features:

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

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

Cisco UCS C220 M5 Rack-Mount Server

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

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

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

Cisco UCS Fabric Extenders

Cisco UCS 2304 Fabric Extender brings the unified fabric into the blade server enclosure, providing multiple 40 Gigabit Ethernet connections between blade servers and the fabric interconnect, simplifying diagnostics, cabling, and management. It is a third-generation I/O Module (IOM) that shares the same form factor as the second-generation Cisco UCS 2200 Series Fabric Extenders and is backward compatible with the shipping Cisco UCS 5108 Blade Server Chassis. The Cisco UCS 2304 connects the I/O fabric between the Cisco UCS 6300 Series Fabric Interconnects and the Cisco UCS 5100 Series Blade Server Chassis, enabling a lossless and deterministic Fibre Channel over Ethernet (FCoE) fabric to connect all blades and chassis together. Because the fabric extender is similar to a distributed line card, it does not perform any switching and is

managed as an extension of the fabric interconnects. This approach reduces the overall infrastructure complexity and enabling Cisco UCS to scale to many chassis without multiplying the number of switches needed, reducing TCO and allowing all chassis to be managed as a single, highly available management domain.

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

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

Cisco VIC Interface Cards

The Cisco UCS Virtual Interface Card (VIC) 1340 is a 2-port 40-Gbps Ethernet or dual 4 x 10-Gbps Ethernet, Fiber Channel over Ethernet (FCoE) capable modular LAN on motherboard (mLOM) designed exclusively for the M4 generation of Cisco UCS B-Series Blade Servers. All the blade servers for both Controllers and Computes will have MLOM VIC 1340 card. Each blade will have a capacity of 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 1385 improves flexibility, performance, and bandwidth for Cisco UCS C-Series Rack Servers. It offers dual-port Enhanced Quad Small Form-Factor Pluggable (QSFP+) 40 Gigabit Ethernet and Fibre Channel over Ethernet (FCoE) in a half-height PCI Express (PCIe) adapter. The 1385 card works with Cisco Nexus 40 Gigabit Ethernet (GE) and 10 GE switches for high-performance applications. The Cisco VIC 1385 implements the Cisco Data Center Virtual Machine Fabric Extender (VM-FEX), which unifies virtual and physical networking into a single infrastructure. The extender provides virtual-machine visibility from the physical network and a consistent network operations model for physical and virtual servers.

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

Cisco UCS 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 which in turn reduce capital and operational expenses of the 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 M5 blade server extends the capabilities **of Cisco’s Unified Computing System portfolio in a half-width blade form factor.** The Cisco UCS B200 M5 harnesses the power of the latest Intel® Xeon® scalable processors product family CPUs with up to 3 TB of RAM– 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.

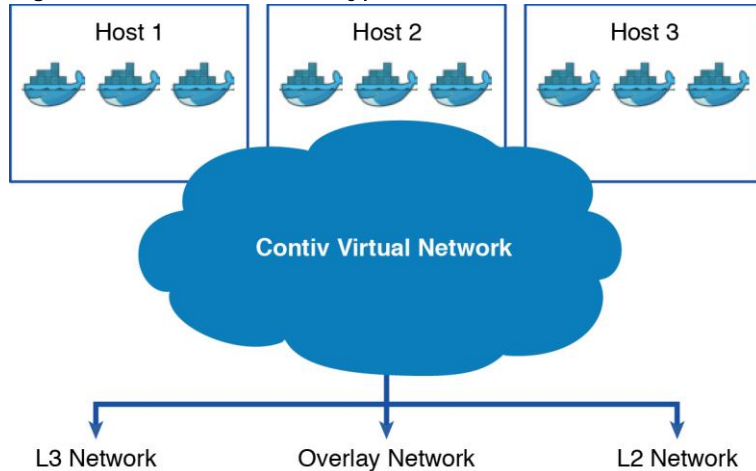
The Cisco Nexus 9000 Series Switches offer both modular and fixed 10/40/100 Gigabit Ethernet switch configurations with scalability up to 30 Tbps of non-blocking performance with less than five-microsecond latency, 1152 x 10 Gbps or 288 x 40 Gbps non-blocking Layer 2 and Layer 3 Ethernet ports and wire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

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

Cisco Contiv

Cisco Contiv delivers policy-based Networking for Containers. Contiv makes it easier to deploy micro-services environment. Contiv provides a higher-level of networking abstraction for microservices and secures application using a rich policy framework. It provides built-in service discovery and service routing for scale out services. With the advent of containers and microservices architecture, there is a need of automated or programmable network infrastructure specifically catering for dynamic workloads which can be formed using containers. Contiv fulfils these requirements in a multi-tenancy model with scale-up and -out architectures with a built-in automated network provisioning.

Figure 2 Basic network types for containers with Contiv



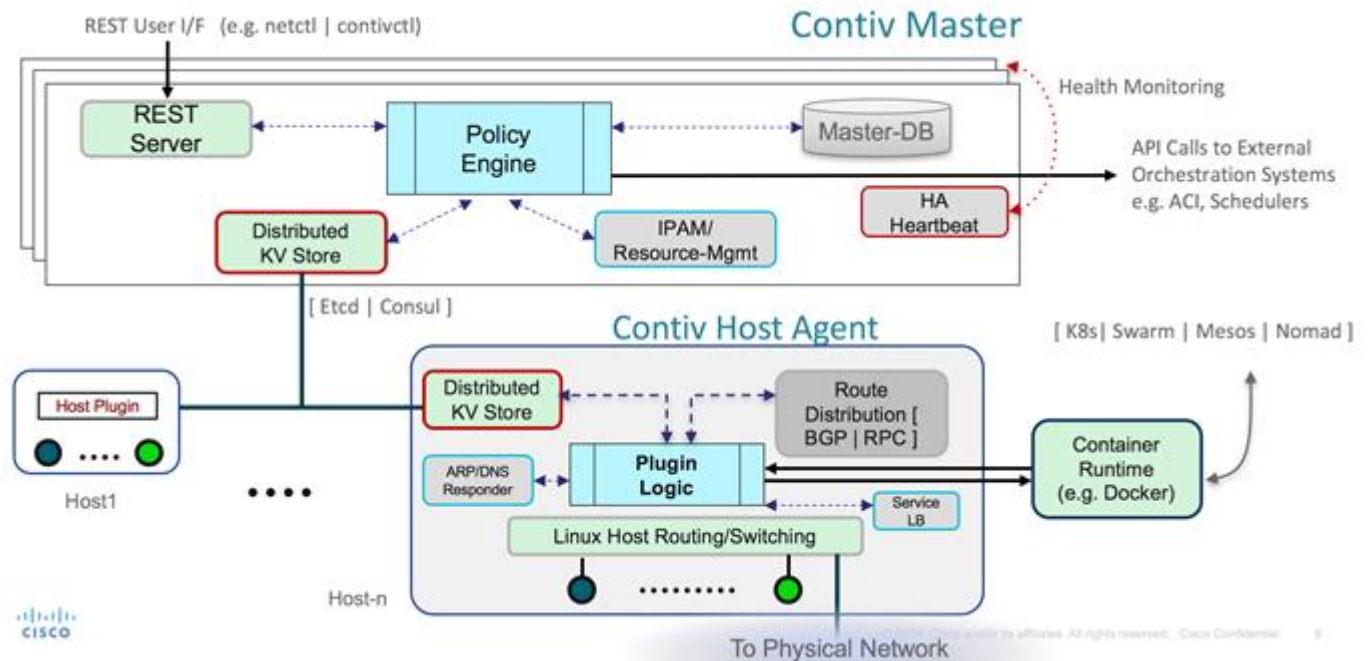
Contiv provides an IP address for each container and eliminates the need for host-based port Network Address Translation (NAT). It works with different types of networks such as pure layer 3 networks, overlay networks, and layer 2 networks, and provides the same virtual network view to containers regardless of the underlying technology. Contiv works with all major schedulers like Kubernetes and Docker Swarm. These schedulers provide compute resources to the containers and Contiv provides networking to them. Contiv supports both CNM (Docker networking Architecture) and CNI (CoreOS, the Kubernetes networking architecture). Contiv supports L2, L3 (BGP), Overlay (VXLAN) and ACI modes. It has built-in east-west service load balancing.

One of the major features of Contiv is to isolate management control plane traffic and application container data traffic. With Cisco VIC in the mix, it becomes easier to provision dedicated physical paths for these traffic types. On top one can apply QOS policies based on their needs of prioritizing traffic types at the hardware level.

Contiv encompasses two major components:

- **Netmaster:** Netmaster is one binary that performs multiple tasks for Contiv. It's a REST API server that can handle multiple requests simultaneously. It learns routes and distributes them to Netplugin nodes. It acts as resource manager which does allocation of IP addresses, VLAN and VXLAN IDs for networks. It uses distributed state store `etcd` to save all the desired runtime for Contiv objects. This makes Contiv completely stateless, scalable, and restart-able. Netmaster has in-built heartbeat mechanism, through which it can talk to peer Netmasters. This avoids the risk of a single point failure. Netmaster can work with external SDN Controllers/policy engines like ACI.
- **Netplugin (Contiv Host Agent):** Each host agent (Netplugin) implements CNI or CNM networking model adopted by popular container orchestration engines like Kubernetes and Docker Swarm. It does communicate with Netmaster over REST API. In addition to this, Contiv uses json-rpc to distribute endpoints from Netplugin to Netmaster. Netplugin handles Up/Down events from Contiv networks and groups. It coordinates with other entities like fetching policies, creating container interface, requesting IP allocation, programming host forwarding. Netplugin uses Contiv's custom open-flow based pipeline on Linux host. It communicates with Open vSwitch (OVS) over the OVS driver. Contiv currently uses OVS for their data path. Plugin architecture of Contiv, makes it very easy to plug in any data path (for example: VPP, BPF etc.).

Figure 3 Contiv architecture diagram showing Netmaster and Netplugin components



Docker Enterprise Edition

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 Enterprise Edition comprises an enterprise container orchestration, application management and enterprise-grade security.

Docker Enterprise Edition (EE) is a Containers-as-a-Service platform for IT that manages and secures diverse applications across disparate infrastructure, both on-premises and in the cloud. Docker EE fuels innovation by bringing traditional applications and microservices built on Windows, Linux or Linux-on-mainframe into a single, secure software supply chain. With Docker, organizations can modernize applications, infrastructure and operational models by bringing forward existing IT investments while integrating new technology at the rate of business.

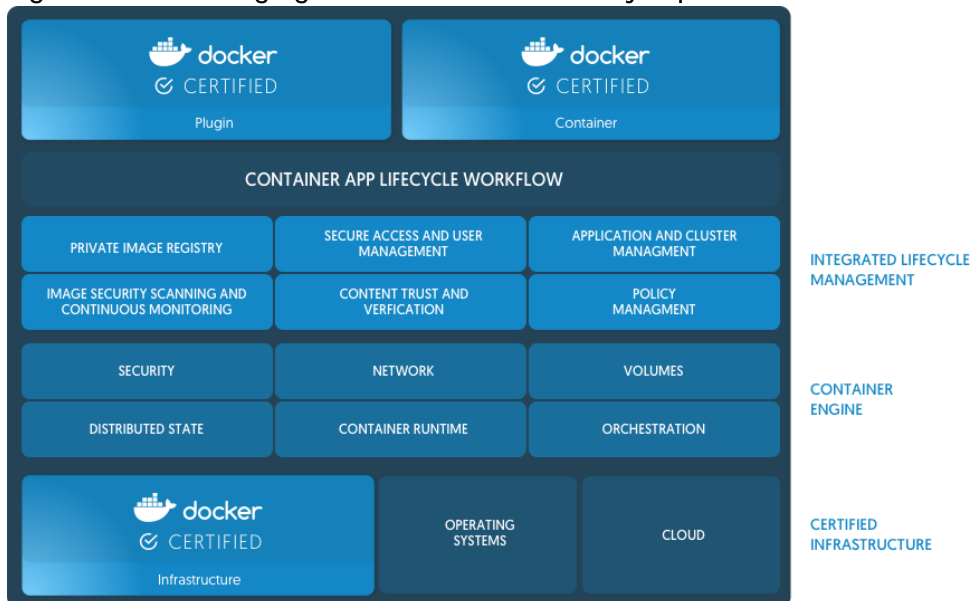
Figure 4 Following figure illustrates high-level Docker Enterprise Edition architecture



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

- Docker Enterprise Edition 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 5 Following figure shows overall security aspects of entire Docker eco-system



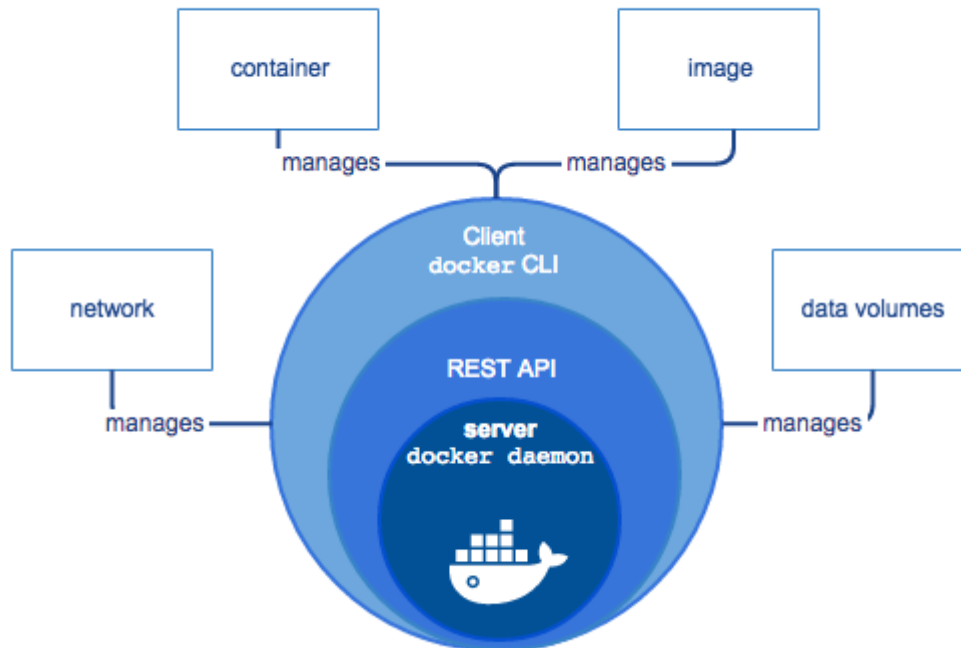
Docker EE Engine

Docker Enterprise Edition engine is the building block for the modern application platform. The Docker EE 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. Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the Docker command).

Following figures illustrate the different Docker components that interact with Docker Engine/ Docker daemon and Docker host

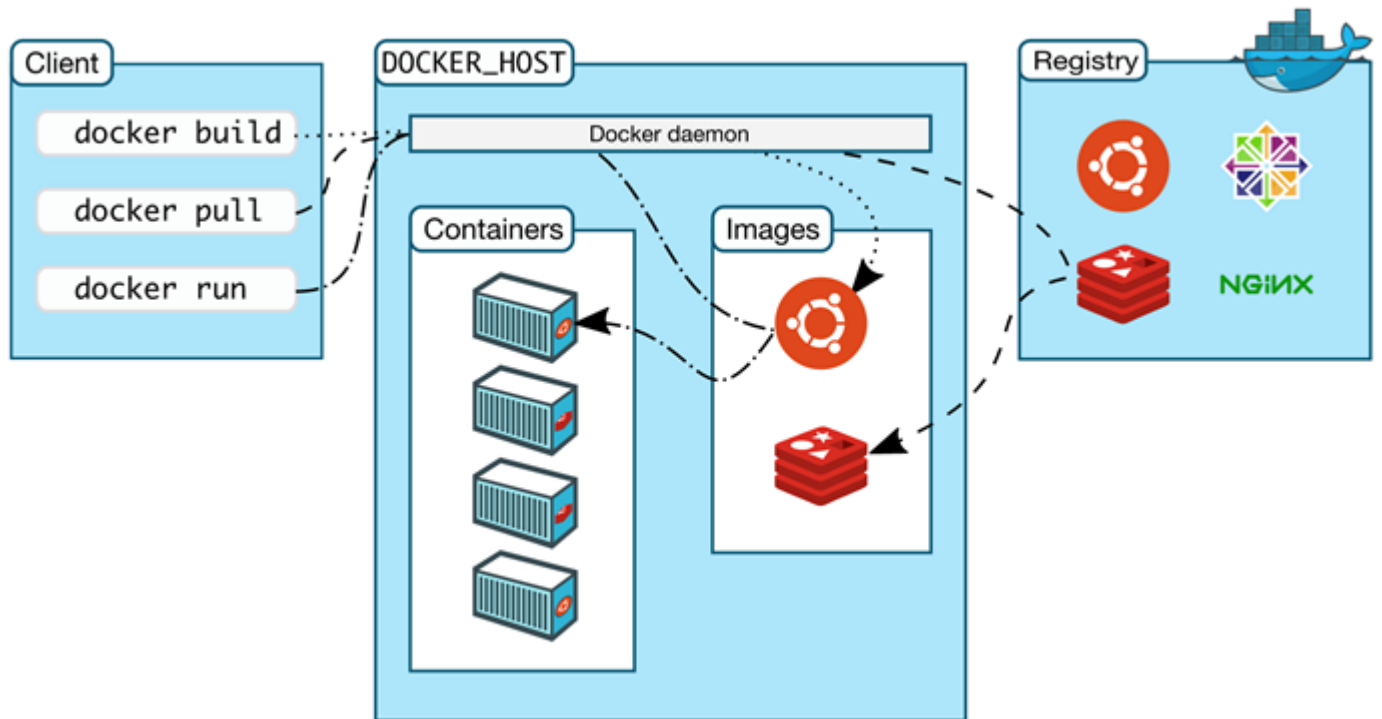
Figure 6 All about Docker Daemon to Docker Host



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.

Docker uses a client-server architecture. Docker client communicates with the Docker daemon, which is responsible for building, running, and distributing the container workloads. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Figure 7 Key Docker Components



Docker Daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker Client

The Docker client is the primary way in which many Docker users interact with Docker. When using commands such as `Docker run` or `Docker service`, the client sends these commands to dockerd, which in turn does the necessary job. The Docker commands use the Docker API. The Docker client can communicate with more than one daemon.

Docker Registries

A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. One can run their own private registry. Docker Enterprise Edition includes Docker Trusted Registry (DTR) by default.

When using the `Docker pull`, `Docker run` and `Docker service` commands, the required images are pulled from configured registry. When using the `Docker push` command, image is pushed to the configured registry.

Docker Objects

Docker container platform essentially comprises a certain components referred to as objects. This section gives a brief overview of these objects.

- Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, Ubuntu image can be used, and Apache web server can be installed along with the configuration details needed to run the application.

You can create your own image or can use the ones that were already created and published in a registry. To build your own image, create a `Dockerfile` with a simple syntax for defining the steps to create and run the image. Note that each instruction in a `Dockerfile` creates a layer in the image. When we change the Dockerfile and rebuild the image, only those layers which have changed get rebuilt. This way the images thus created are lightweight, small, and fast, when compared to other virtualization technologies.

- Containers

A container is a runnable instance of an image. One can create, start, stop, move, or delete a container using the Docker API or CLI. One can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can **control how isolated a container's network, storage, or other underlying subsystems** are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

- Services

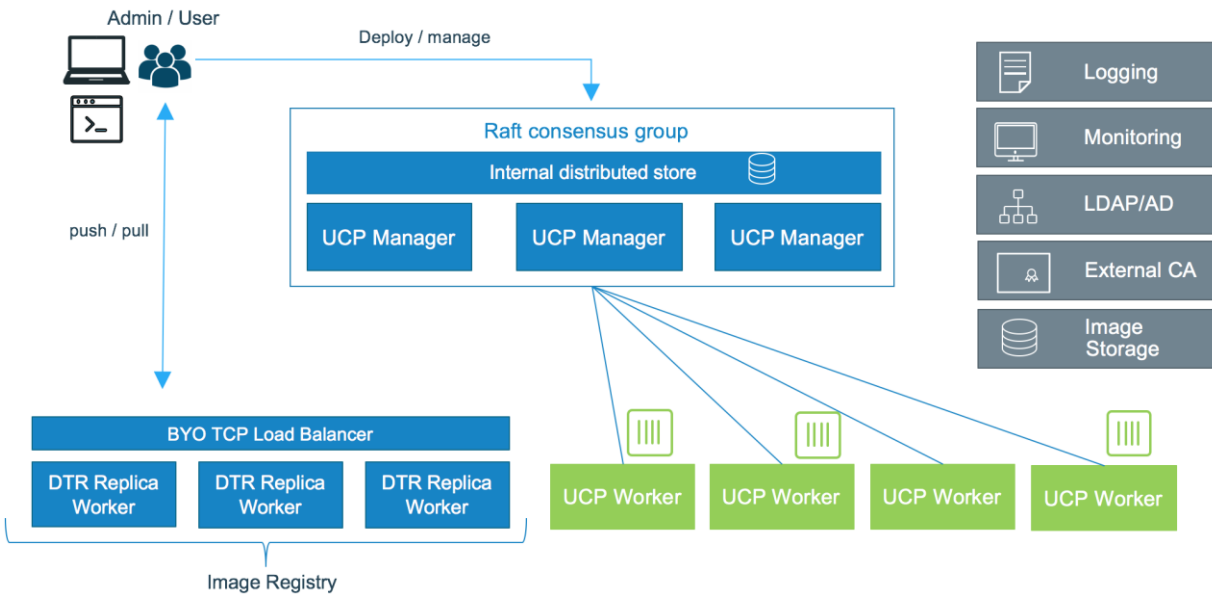
Services allow us to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and the daemons all communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application.

Docker Universal Control Plane (UCP)

Universal Control Plane is a containerized application that runs on Docker Enterprise Edition and extends its functionality to make it easier to deploy, configure, and monitor your applications at scale. UCP also secures Docker with role-based access control so that only authorized users can make changes and deploy applications to your Docker cluster. It 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.

Once Universal Control Plane (UCP) instance is deployed, development and IT operations no longer interact with Docker Engine directly, but interact with UCP instead. Since the Docker UCP exposes the standard Docker API transparently, you can use the tools like the Docker CLI client and Docker Compose. Docker UCP leverages the clustering and orchestration functionality provided by Docker.

Figure 8 High level architecture of Docker UCP

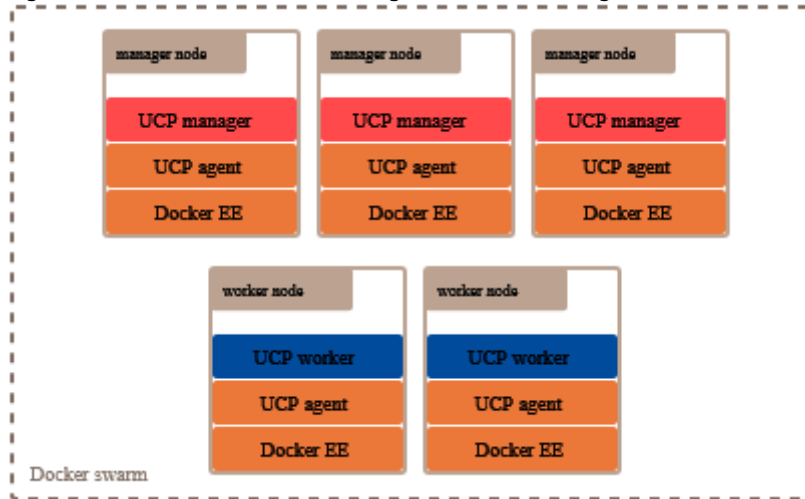


A swarm is a collection of nodes that are in the same Docker cluster. Nodes in a Docker swarm operate in one of two modes: Manager or Worker. If nodes are not already running in a swarm when installing UCP, nodes will be configured to run in swarm mode. When Docker UCP gets deployed, the globally scheduled service called `ucp-agent` is executed. **This service monitors the node where it's running and starts and stops the UCP services, based on whether the node is a manager or a worker node.**

- **Manager:** the `ucp-agent` service automatically starts serving all UCP components, including the Docker UCP web UI and data stores used by UCP. The `ucp-agent` accomplishes this by deploying several containers on the node. By promoting a node to manager, Docker UCP automatically becomes highly available and fault tolerant.
- **Worker:** on worker nodes, the `ucp-agent` service starts serving a proxy service that ensures only authorized users and other UCP services can run Docker commands on the node. The `ucp-agent` deploys a subset of containers on worker nodes.

Docker UCP administrators have a choice of restricting the scheduling of application containers on manager nodes only. By default all the nodes irrespective of their roles, gets the application containers scheduled based on resource availability.

Figure 9 Docker UCP leverages the clustering and orchestration functionality provided by Docker



Docker UCP Internal Components

The core component of UCP is a globally-scheduled service called `ucp-agent`. Installing UCP on a node, or **join a node to a swarm that's being managed by UCP**, the `ucp-agent` service starts running on that node. Once this service starts, it deploys containers with other UCP components, and it ensures they keep running. The UCP components that are deployed on a node depend on whether the node is a manager or worker.

Table 1 These are the UCP services running on manager nodes

Name	Description
<code>ucp-agent</code>	Monitors the node and ensures the right UCP services are running
<code>ucp-controller</code>	The UCP web server
<code>ucp-swarm-manager</code>	Used to provide backwards-compatibility with Docker Swarm
<code>ucp-reconcile</code>	When <code>ucp-agent</code> detects that the node is not running the right UCP components, it starts the <code>ucp-reconcile</code> container to converge the node to its desired state. It is expected for the <code>ucp-reconcile</code> container to remain in an exited state when the node is healthy.
<code>ucp-auth-api</code>	The centralized API for identity and authentication used by UCP and DTR
<code>ucp-auth-worker</code>	Performs scheduled LDAP synchronizations and cleans data on the <code>ucp-auth-store</code>
<code>ucp-auth-store</code>	Stores authentication configurations, and data for users, organizations, and teams
<code>ucp-kv</code>	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 used for TLS communication between UCP components
ucp-client-root-ca	A certificate authority to sign user bundles
ucp-dsinfo	Docker system information script to assist in troubleshooting
ucp-metrics	Used to collect and process metrics for a node, like the disk space available
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine to UCP components.

UCP Components in worker nodes – Worker nodes are dedicated for taking application workloads.

Table 2 Docker UCP services running on worker nodes

Name	Description
ucp-agent	Monitors the node and ensures the right UCP services are running
ucp-reconcile	When ucp-agent detects that the node is not running the right UCP components, it starts the ucp-reconcile container to converge the node to its desired state. It is expected for the ucp-reconcile container to remain in an exited state when the node is healthy.
ucp-dsinfo	Docker system information script to assist in troubleshooting
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine to UCP components.

Docker UCP Storage Volume Usage

While installing Docker UCP storage volumes get created with default volume driver. Volume drivers used for these volumes can be customized, by creating the volumes before installing UCP. In this solution, Docker device-mapper driver in direct-lvm mode is used for creating volumes. During installation, Docker UCP checks for the volumes that do not exist on the node, and creates them using the default volume driver. By default, the data for these volumes can be found at ``/var/lib/docker/volumes/<volume_name>/_data``.

Table 3 Docker UCP uses these named volumes for providing data persistency

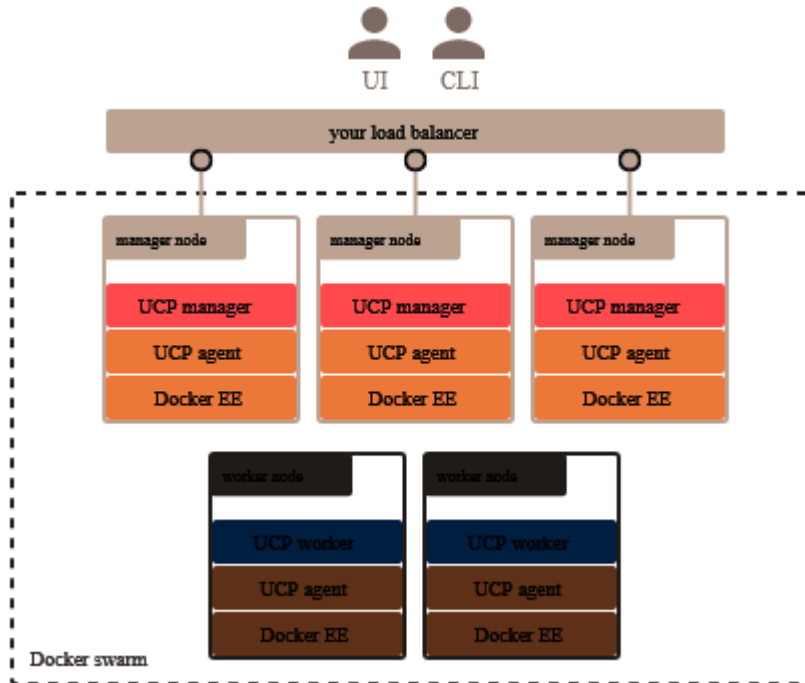
Volume name	Description
ucp-auth-api-certs	Certificate and keys for the authentication and authorization service
ucp-auth-store-certs	Certificate and keys for the authentication and authorization store
ucp-auth-store-	Data of the authentication and authorization store,

data	replicated across managers
ucp-auth-worker-certs	Certificate and keys for authentication worker
ucp-auth-worker-data	Data of the authentication worker
ucp-metrics-data	Monitoring data gathered by UCP
ucp-metrics-inventory	Configuration file used by the ucp-metrics service
ucp-client-root-ca	Root key material for the UCP root CA that issues client certificates
ucp-cluster-root-ca	Root key material for the UCP root CA that issues certificates for swarm members
ucp-controller-client-certs	Certificate and keys used by the UCP web server to communicate with other UCP components
ucp-controller-server-certs	Certificate and keys for the UCP web server running in the node
ucp-kv	UCP configuration data, replicated across managers
ucp-kv-certs	Certificates and keys for the key-value store
ucp-node-certs	Certificate and keys for node communication

User Interaction with Docker UCP

There are two ways to interact with UCP: the web UI and the CLI. You can use the UCP web UI to manage your swarm, grant and revoke user permissions, deploy, configure, manage, and monitor your applications. UCP also exposes the standard Docker API, so that you can continue using the existing tools like the Docker CLI client. Since UCP secures your cluster with role-based access control, you need to configure your Docker CLI client and other client tools to authenticate your requests using client certificates that you can download from the UCP profile page.

Figure 10 User interface with Docker UCP



Docker Universal Control Plane works in high-availability mode. Adding replicas to first Manager node makes the cluster HA ready. A minimum three-node 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 external load-balancer for its management services. It needs external load-balancer to balance user requests across all master nodes for UCP and/or DTR access in highly-available mode.



In this solution, `ha-proxy`, a Linux based software external load-balancer is used to validate high-availability of UCP, DTR and Contiv UI services. Example `ha-proxy` configuration is shown in the appendix section in this document. You can choose any external software/hardware load-balancer of your choice which meets Docker configuration prerequisites.

Docker Trusted Registry (DTR)

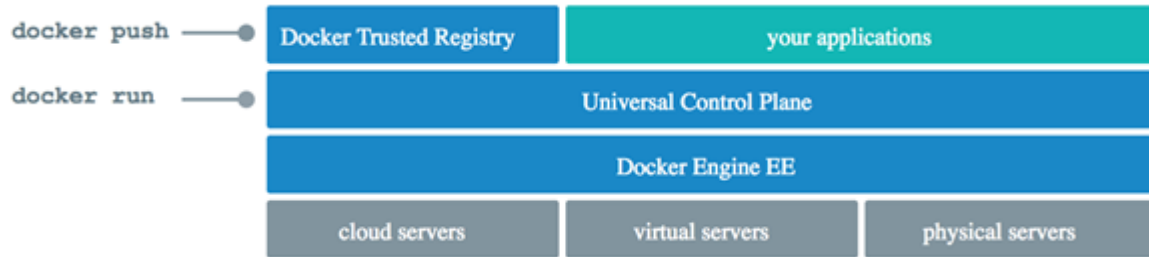
Docker Trusted Registry (DTR) is the enterprise-grade image storage solution from Docker. Installation can be done behind firewall so that it can securely store and manage the Docker images for the use in applications. DTR can be installed on-premises, or on a virtual private cloud. And with it, one can store Docker images securely, behind the firewall. DTR can be used as part of continuous integration, and continuous delivery processes to build, ship, and run applications. DTR has a web based user interface that allows authorized users from the organization to browse Docker images. It provides information about who pushed what image at what time. It even allows you to see what `dockerfile` lines were used to produce the image and, if security scanning is enabled, to see a list of all of the software installed in your images.

DTR is highly available through the use of multiple replicas of all containers and metadata such that if a machine fails, DTR continues to operate and can be repaired. DTR has the ability to cache images closer to

users to reduce the amount of bandwidth used during Docker pulls. DTR has the ability to clean up unreferenced manifests and layers.

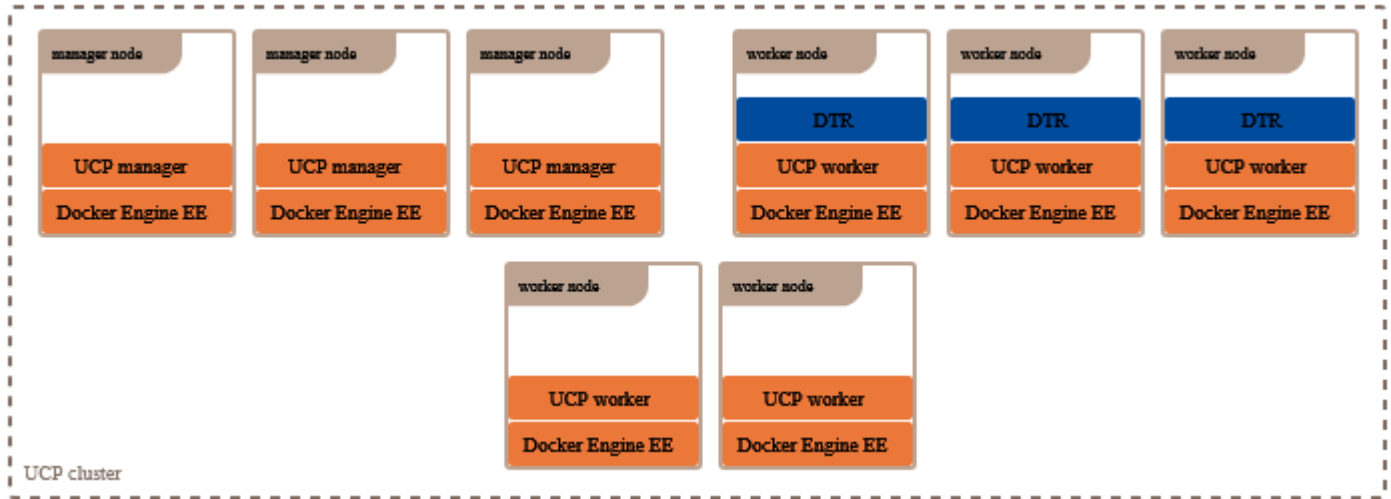
DTR uses the same authentication mechanism as Docker Universal Control Plane. Users can be managed manually or synched from LDAP or Active Directory. DTR uses Role Based Access Control (RBAC) to allow implementing fine-grained access control policies for who has access to Docker images.

Figure 11 Docker Trusted Registry (DTR) – a containerized application on Docker UCP cluster



For high-availability, DTR deployment is done with a minimum 3 replicas, one on each UCP worker nodes. All DTR replicas run the same set of services and the configuration changes are automatically propagated to other DTR replicas.

Figure 12 Highly-available Docker EE architecture showing infrastructure services distributed across cluster nodes



Docker DTR Internal Components

Following application containers gets deployed on installing DTR.

Table 4 DTR service application containers and their functional roles

Name	Description
dtr-api-<replica_id>	Execute the DTR business logic. It serves the DTR web application and API
dtr-garant-<replica_id>	Manages DTR authentication
dtr-jobrunner-<replica_id>	Runs the cleanup jobs in the back-ground

dtr-nginx-<replica_id>	Receives HTTP and HTTPS requests and proxies them to other DTR components. Default listening ports are 80 and 443 on the host
dtr-notary-server-<replica_id>	Receives, validates, and serves content trust metadata, and is consulted when pushing or pulling to DTR with content trust enabled
dtr-notary-signer-<replica_id>	Performs server-side timestamp and snapshot signing for content trust metadata
dtr-registry-<replica_id>	Implements the functionality for pulling and pushing Docker images. It also handles how images are stored
dtr-rethinkdb-<replica_id>	A database of persisting repository metadata
dtr-scanningstore-<replica_id>	Stores security scanning data

DTR service application containers communicate with each other on cluster nodes via `dtr-ol` network. This overlay network is created as part of DTR installation and allows DTR data replication with the DTR cluster nodes.

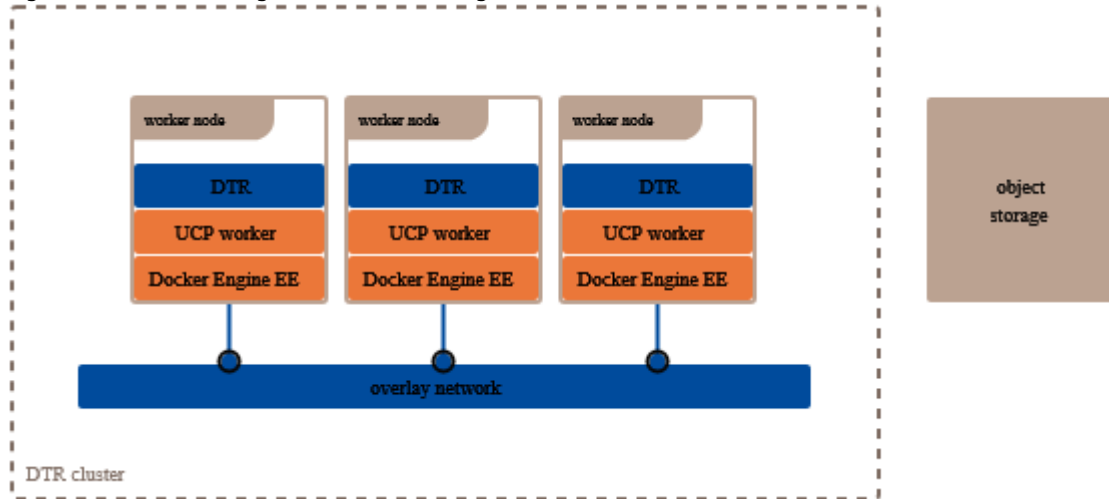
Docker UCP Storage Volume Usage

Table 5 List of named volumes for DTR data persistency

Volume name	Description
dtr-ca-<replica_id>	Root key material for the DTR root CA that issues certificates
dtr-notary-<replica_id>	Certificates and keys for the Notary components
dtr-postgress-<replica_id>	Vulnerability scan data
dtr-registry-<replica_id>	Docker images data, if DTR is configured to store images on the local filesystem
dtr-rethink-<replica_id>	Repository metadata
dtr-nfs-registry-<replica_id>	Docker images data, if DTR is configured to store images on NFS

By default, Docker Trusted Registry stores images on the filesystem of the node where it is running, but for the enterprise grade deployment it is recommended to be configured with a centralized storage backend.

Figure 13 DTR Integration with storage backend



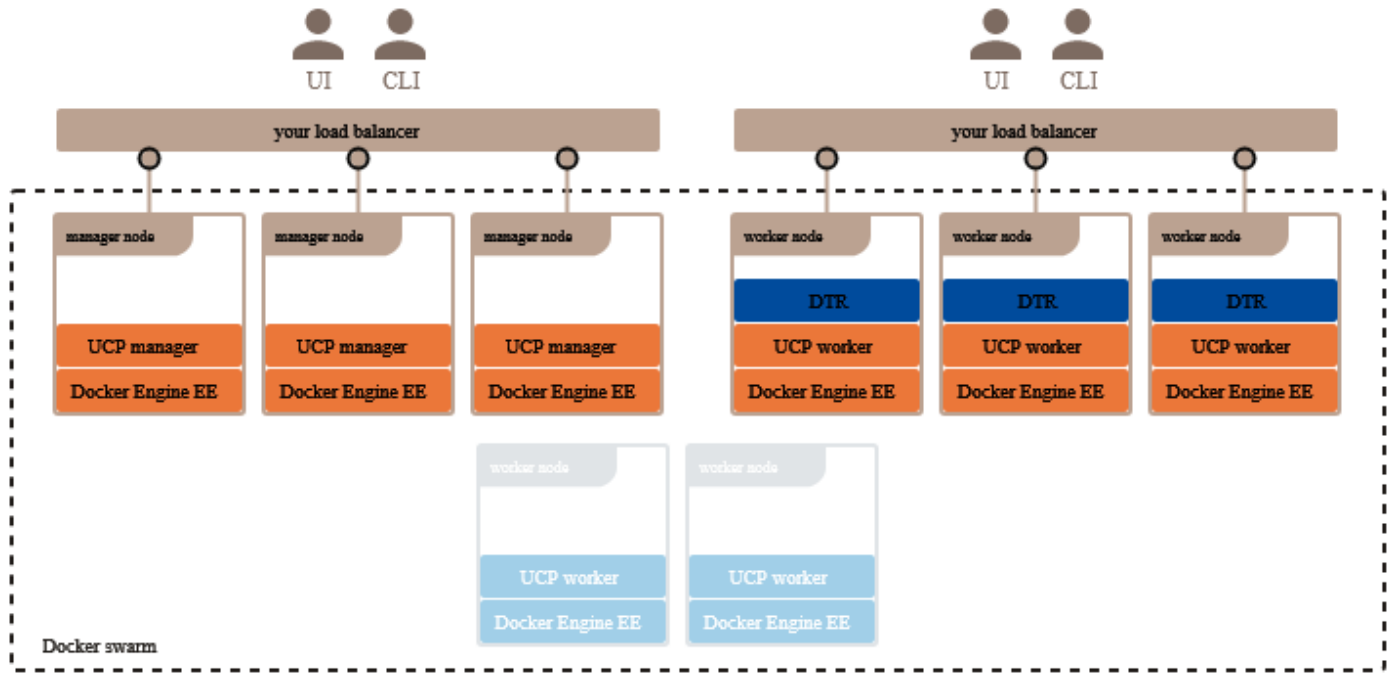
DTR supports following storage back-ends:

- NFS
- Amazon S3
- Cleversafe
- Google Cloud Storage
- OpenStack Swift
- Microsoft Azure

User Interfaces with Docker DTR

DTR has a web UI which gets installed and configured as part of DTR installation. DTR UI can be accessed from any of the DTR nodes and presents same repository data, irrespective of from which node its being accessed. In order to achieve highly available access, DTR can be front-ended with an external load-balancer, same as in the case of Docker UCP. Users can push and pull images using standard Docker cli client or other tools.

Figure 14 User interface with Docker UCP and DTR



In this solution, 'ha-proxy', a Linux based software external load-balancer is used to validate high-availability of UCP, DTR and Contiv UI services. Example 'ha-proxy' configuration is shown in the appendix section in this document. You can choose the external software/hardware load-balancer of your choice which meets Docker configuration prerequisites.

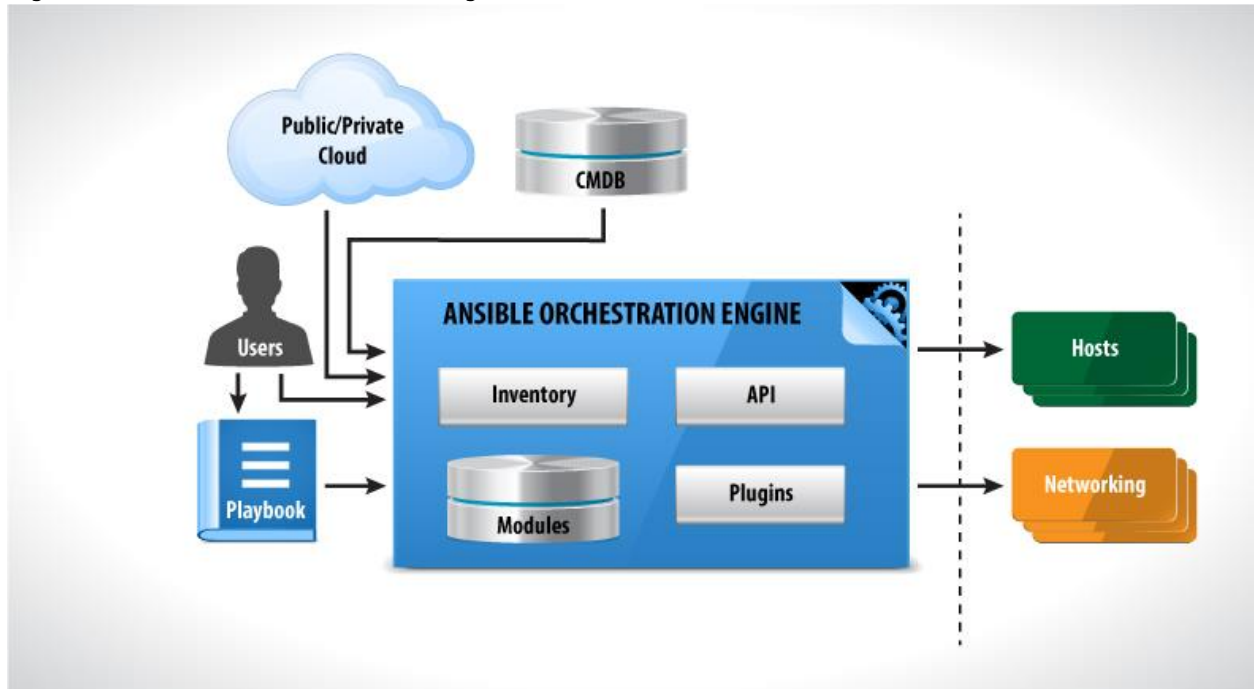
Ansible

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

Designed for multi-tier deployments since day one, Ansible models your IT infrastructure by orchestrating all the systems rather than just managing one system at a time.

Ansible is a simple automation language that can **perfectly describe an IT application infrastructure**. It's easy-to-learn, self-documenting, and uses no agents, no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs. **It's one of the simplest and the easiest to get started with because it's "just SSH"; It uses SSH to connect to servers and run the configured Tasks.**

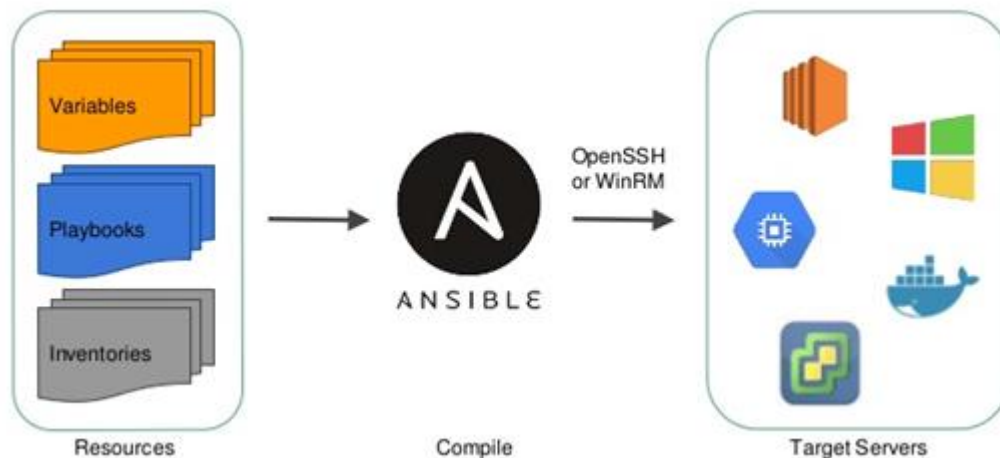
Figure 15 Ansible Orchestration Engine



Ansible works by connecting to the nodes through small programs, called "Ansible modules". These programs are written to be resource models of the desired state of the system. Ansible executes these modules over SSH by default.

Your library of modules can reside on any machine, and there are no servers, daemons, or databases required. Typically, you work with the favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content.

Figure 16 Ansible automation engine



By default, Ansible represents what machines it manages using a very simple INI file that puts all of managed machines in groups of our own choosing. Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called 'group_vars/' or 'host_vars/') or directly in the inventory file.

Ansible is from Red Hat and becoming a de facto open source automation technology, and is available through Extra Packages for Enterprise Linux (EPEL).

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 section provides information on the architectural details of Cisco UCS infrastructure, Cisco Contiv and Docker Enterprise Edition. This section also illustrates the software and hardware solution components and discusses the details on obtaining the optimal, high-performance and highly-available infrastructure for creating application containers and microservices in production environment.

Cisco UCS and Docker Enterprise Edition

This solution consists of two alternate topologies, targeted for production and dev/test deployments. For production environment, Cisco UCS B200 M5 Blade Servers are leveraged, with nodes dedicated for Docker Universal Control Plane (UCP) and Docker Trusted Registry (DTR) services within Docker Enterprise Edition. This document also provides a second architecture with a minimal number of servers to implement Docker Enterprise Edition on Cisco UCS using C220 M5 Rack-Mount Servers. The second alternative (referred to as second architecture in this solution) is a best fit for typical dev/test deployment environments. Cisco UCS Fabric Interconnects and Cisco Nexus TOR switches are used in both these topologies. The Docker Enterprise Edition runs on bare metal nodes running Red Hat Enterprise Linux Operating System. Cisco UCS servers provide highly-available hardware platform centrally managed by Cisco UCS Manager Software residing on Cisco Fabric Interconnects. As an important component of the Docker Enterprise Edition (Docker EE), Docker Universal Control Plane (Docker UCP) provides the redundancy and high-availability of the Docker EE Engines and management control plane. This solution holistically offers container management for diverse application environment to be deployed in DevOps and Production environments.

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 UCP services are spread over three B-Series servers for providing management control plane redundancy and high-availability; also, DTR services are configured to run on three dedicated B-Series servers for providing high-availability to the image repository.

In the second architecture Docker UCP and DTR services are co-hosted on three C-Series rack servers. Docker UCP administrator can make a choice to allow distribution of application container workload to be spread across the Docker UCP manager nodes as well along with the worker nodes.

Both Docker UCP and DTR dashboards require an external load balancer to access management interfaces to make them operate in a highly available mode. External load balancer provides virtual IP address to front-end the UCP and DTR management dashboards separately. External load balancer is out of the scope of this document **as the load balancer is purely a customer's choice and configuration** follows a standard procedure.



In this solution, 'ha-proxy', a Linux based software external load-balancer is used to validate high-availability of UCP, DTR and Contiv UI services. Example 'ha-proxy' configuration is shown in the appendix

section in this document. You can choose the external software/hardware load-balancer of your choice which meets Docker configuration prerequisites.

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

Cisco Contiv

Contiv provides networking to containers natively to the physical network. It supports both major networking models:

- The libnetwork/Container Network Model (CNM)
- The Container Network Interface (CNI)

Contiv provides a pluggable networking option to Docker and Kubernetes ecosystem. In this solution, we have used Contiv network driver to work with Docker EE cluster in native Swarm mode. It gets plugged in to the libnetwork/Container Network Model (CNM).

Container **Network Model is Docker's libnetwork container network project. Salient features of this model** are listed below:

- A network is a collection of arbitrary endpoints
- **An endpoint is a container's interface into a network**
- A container can belong to multiple endpoints and therefore multiple networks
- CNM allows for co-existence of multiple drivers each managing their own networks
- Network driver APIs allow to create/delete network, create/delete/join/leave endpoints
- IPAM Driver APIs allow to create/delete pool, Allocate/Free IP Addresses

Contiv has L2, L3 (BGP), Overlay (VXLAN) and ACI modes. It has built in east-west service load balancing. Contiv also provides traffic isolation through control and data traffic. Contiv provides an IP address per container and eliminates the need for host-based port NAT. Since it gives native networking capabilities to the containers, it can work with different types of physical networks like pure layer 3 networks, overlay networks, and layer 2 networks and provides the same virtual network view to containers regardless of the underlying technology.

This solution recommends using Contiv Network driver plugin in L2 VLAN - Bridge forwarding mode. Contiv L2 VLAN mode addresses following use cases:

- Native/Underlay visibility - troubleshooting and monitoring easy with legacy tool sets
- No changes needed - in the existing well-understood and good-old classic topology
- Easy migration possibilities from BM/VM to containers - works without configuration, topology and existing security policies

- Multi-Tenancy – top level object to achieve Data-path isolation and integration for external connectivity

Solution lays emphasis on using Contiv Network driver for container networking by following high level steps as given below:

- A VLAN or small set of VLANs pre-configured once on physical devices for the containers
- Each container gets an IP, which is accessible from anywhere
- ARP broadcast get suppressed at the host level
- Configurations of static SVI, VLANs and VPC on physical devices
- Contiv host agent applies the policy on the host

A typical workflow is as described below:

1. Configure switched virtual interfaces (SVIs) on switches
2. Create VLAN networks with subnets and gateways
3. Start containers on networks created on hosts using Contiv
4. Verify IP address, routes, and connectivity between containers

Physical Topology

This solution is designed and proposes two separate topologies based on Docker EE production as well as dev/test deployment requirements for the Enterprise. The two topologies make use of Cisco UCS server hardware form factor optimally. Where there is a need for higher local storage and compute resource to be made available for application containers UCS C-series server topology fits best. It can either be used for high density dev/test environment or even for production use cases, where the need for scaling up the environment are less and as such Docker EE infra services can be co-hosted on controller nodes and can take up the application container workloads. Cisco UCS B-series topology fits best for the use cases where there is a strong need to scale out the environment in future and Docker EE infra services are to be hosted separately. This topology is a best fit for application container environment for production use cases.

The following figures illustrate the two types of reference architectures and the physical back-end connectivity of the hardware components:

Figure 17 Physical Topology – First Architecture

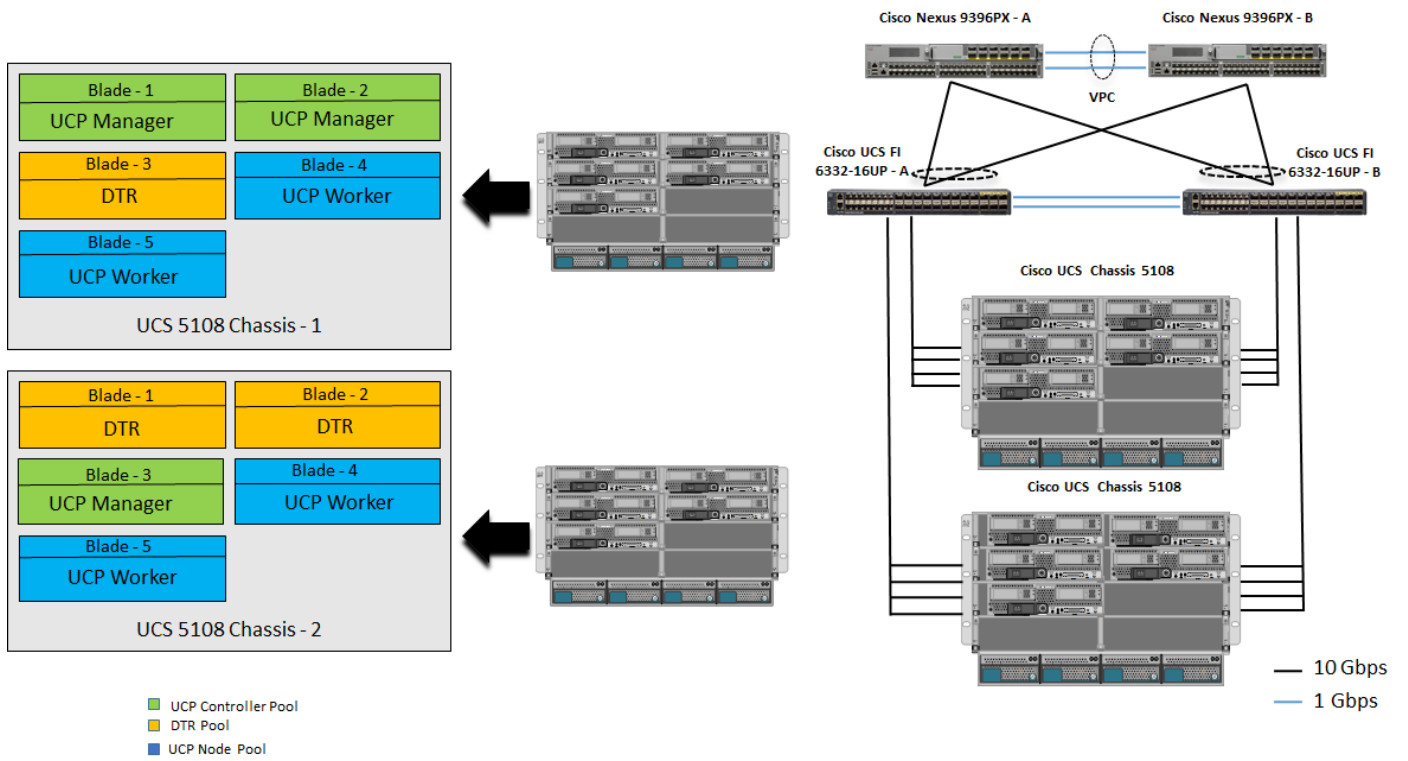


Figure 18 Cabling Diagram – First Architecture

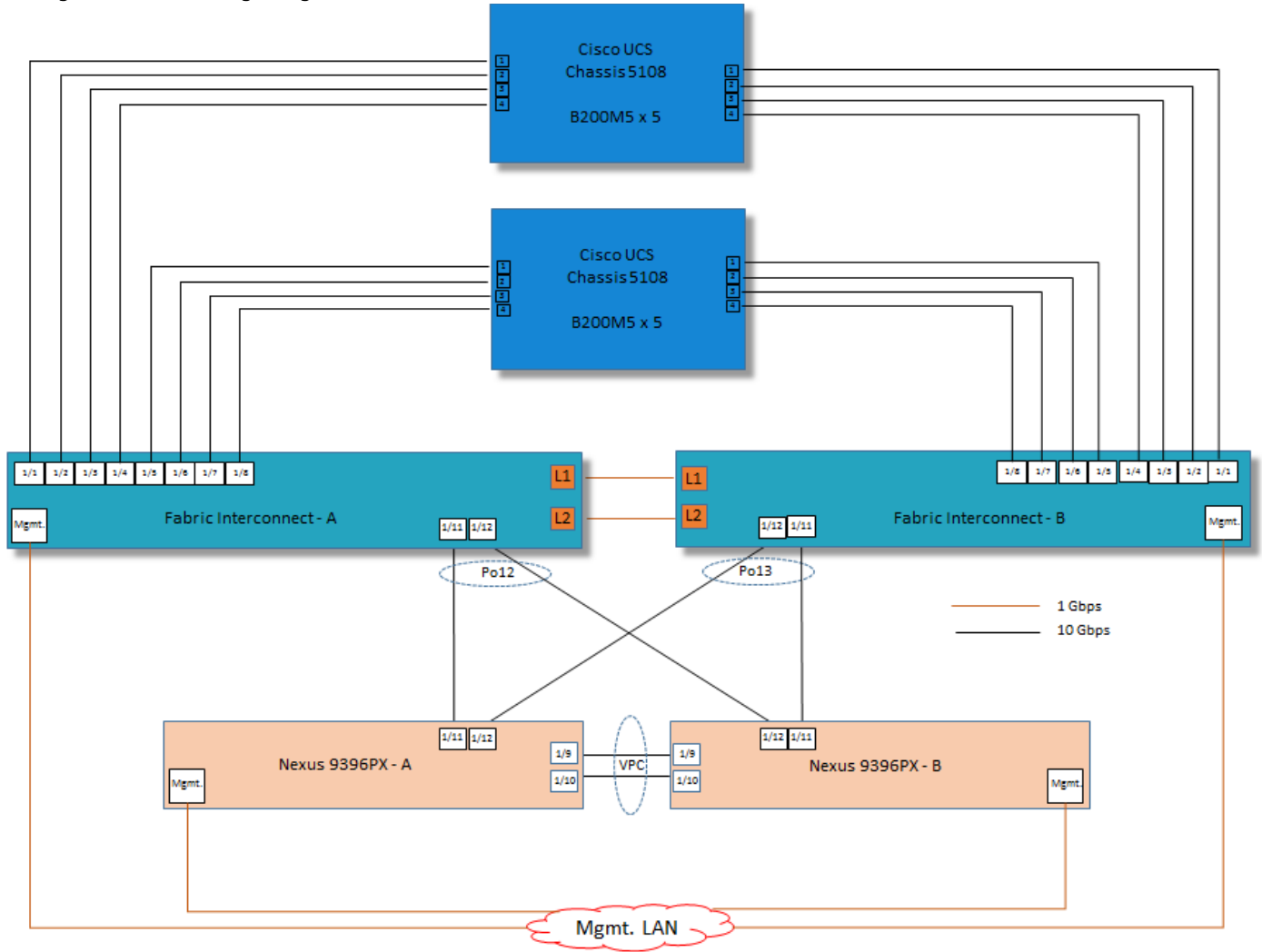
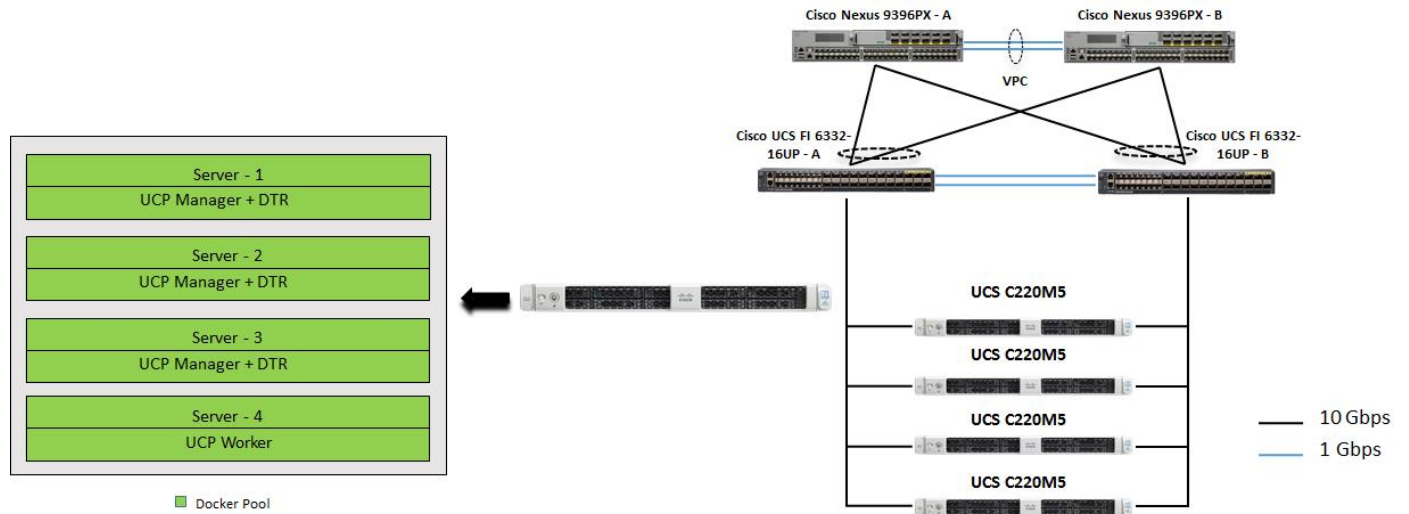


Figure 19 Physical Topology – Second Architecture

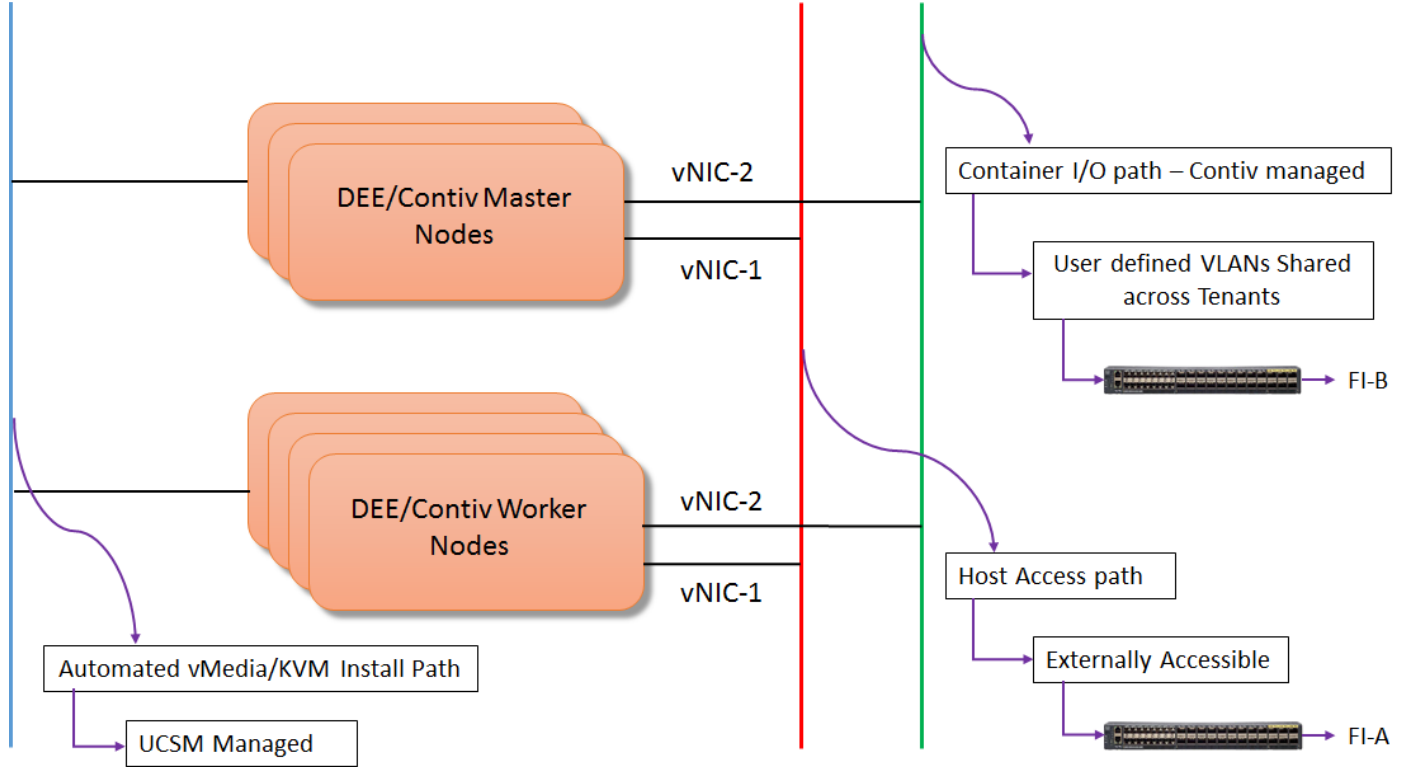


Logical Topology

This section provides details on how Cisco VIC with Contiv enables application containers to use dedicated I/O path for network access in a secured environment with multi-tenancy. With Contiv and Cisco VIC, containers get a dedicated physical path to a classic L2 VLAN topology with better line rate efficiency.

This solution focuses on Contiv L2 VLAN forwarding mode in a multi-tenant environment. Contiv takes advantage of Cisco VIC to segregate container i/o from the management/control plane traffic. It provides dedicated physical path for container application network. To further enhance the value of Cisco UCS by optimizing the infrastructure utilization, the traffic paths were configured to segregate all management/control traffic through fabric interconnect A, and container data traffic through fabric interconnect B.

Figure 20 Docker Enterprise Edition cluster with Contiv – Network layout per host



Following Figures 22 through 26 explain data path isolation for multi-tenant and L2 VLAN packet flow use cases.

Figure 21 Data path isolation between multi-tenants with Contiv

Host

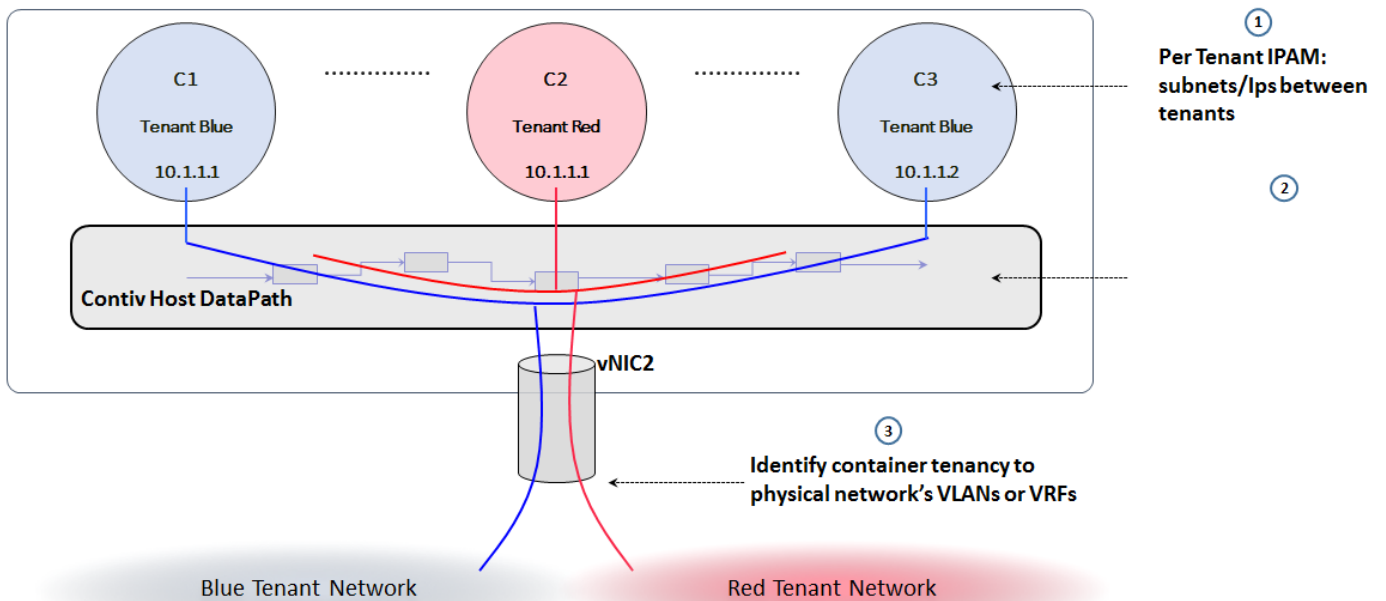


Figure 22 Case-1: Data Packet from one container to another container within same host

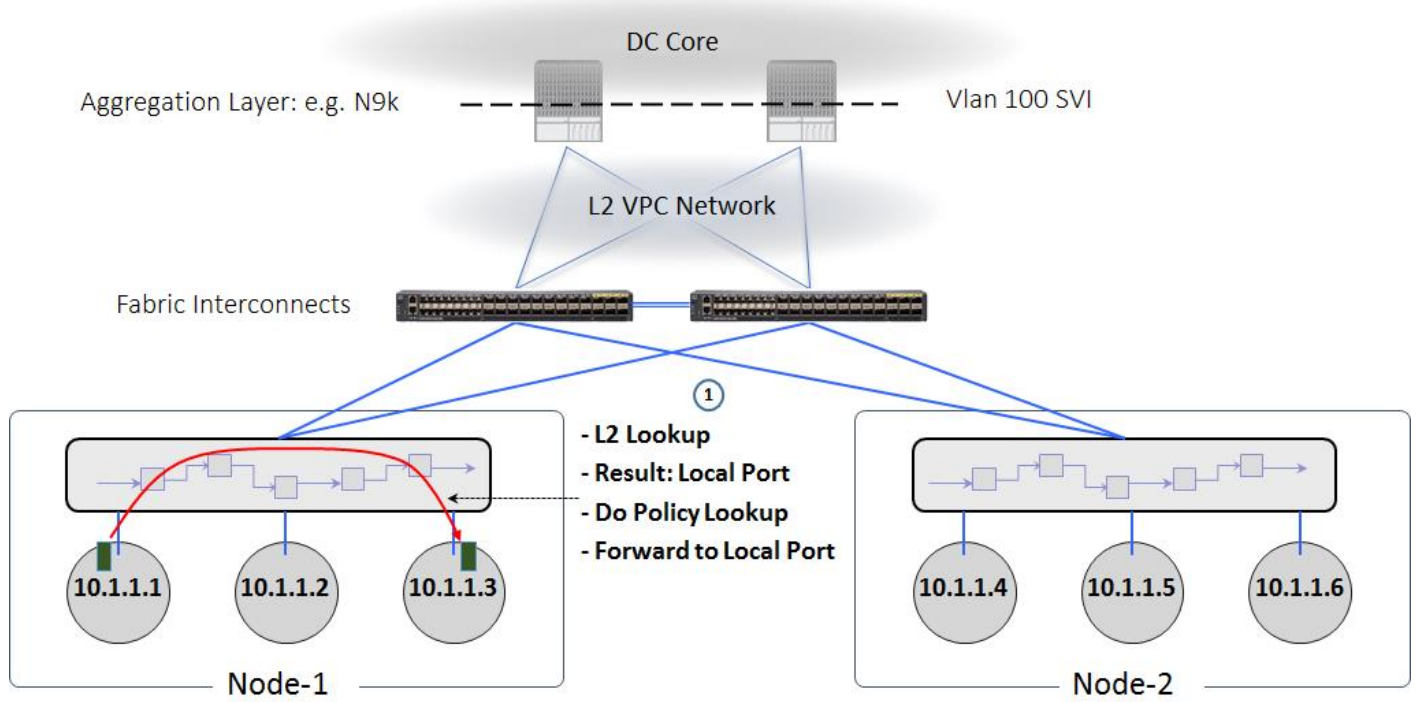


Figure 23 Case-2: ARP Requests from a container to any other container's IP within cluster

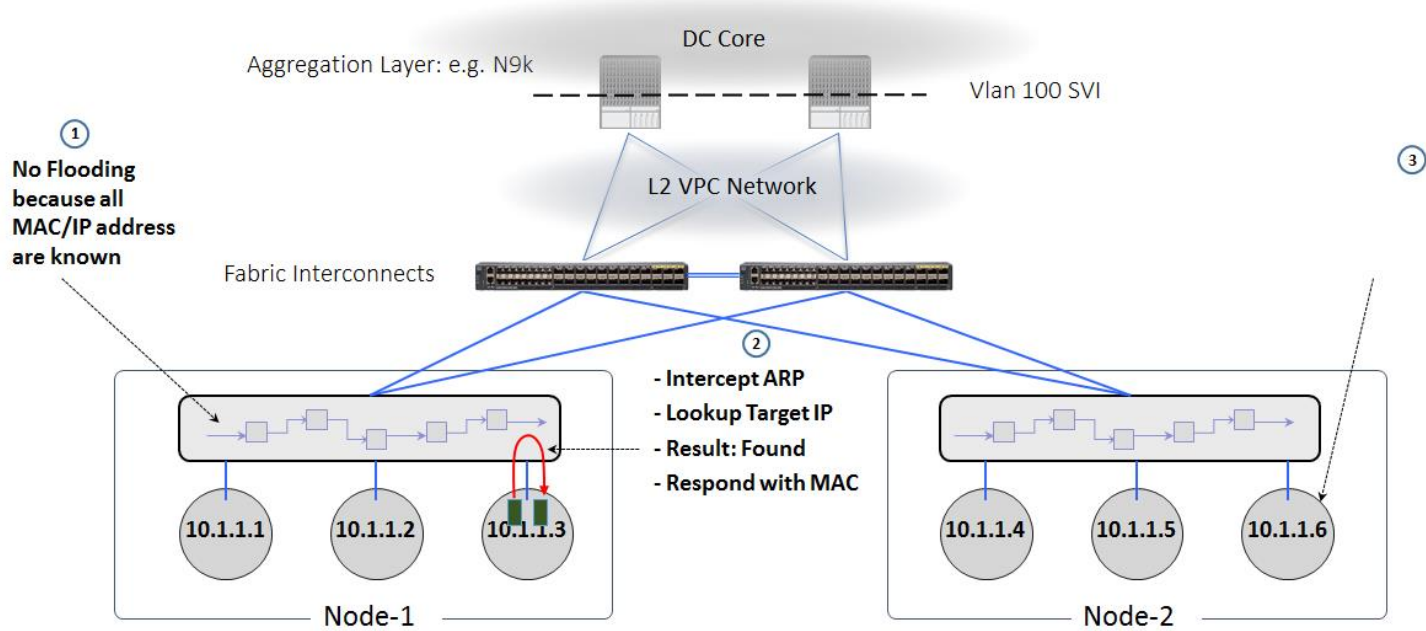


Figure 24 Case-3: Data packet from one container to another container on different host

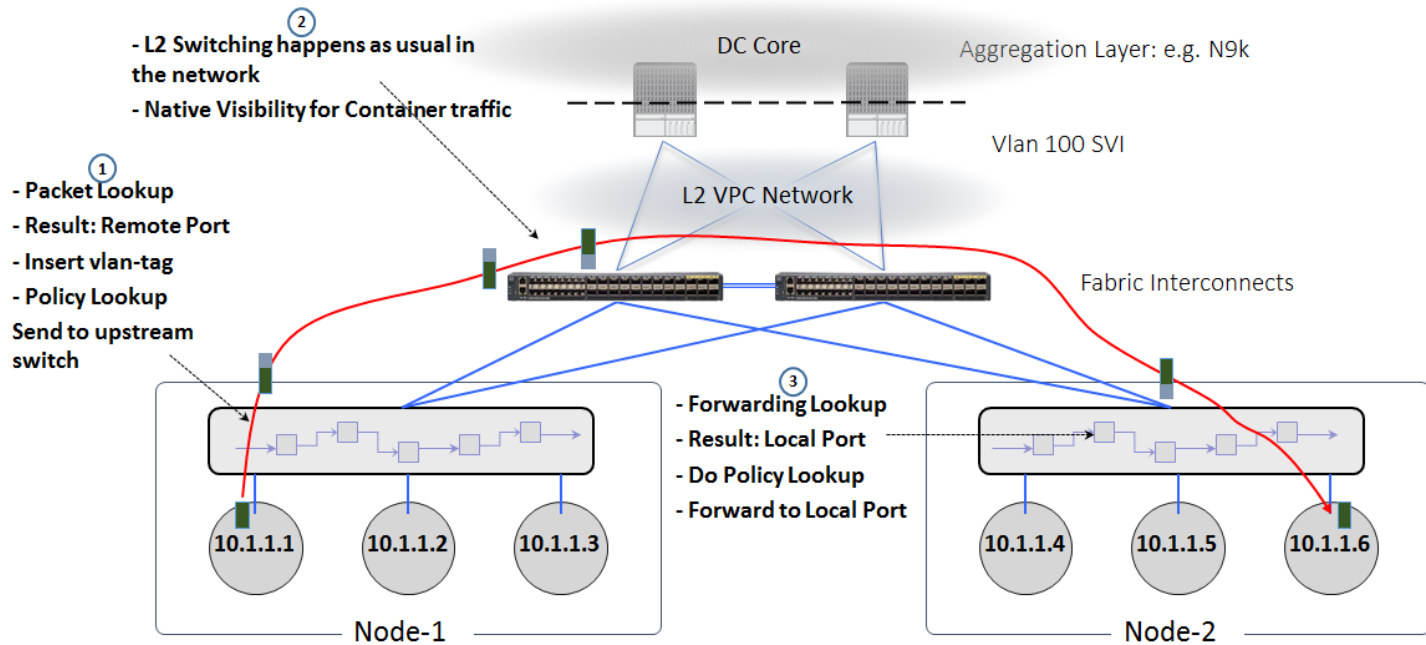
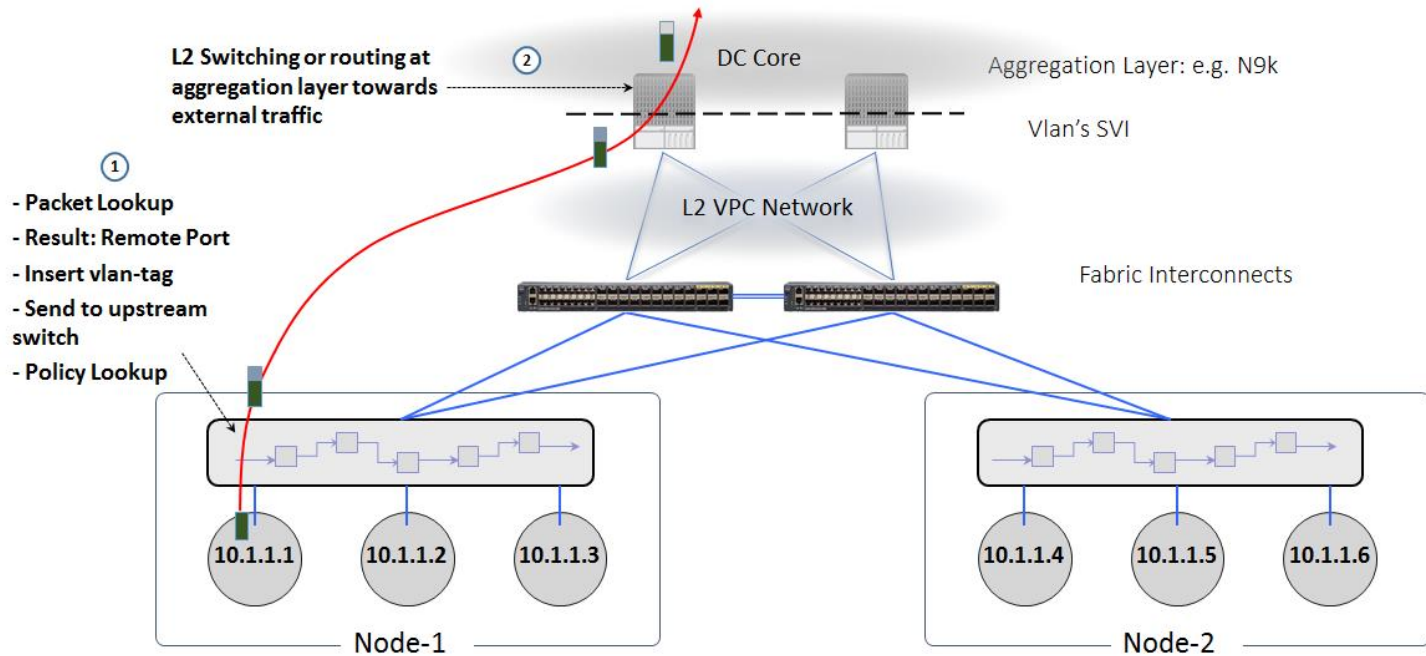


Figure 25 Case-4: Data packet to/from container to outside



Ansible Playbook

In this solution Ansible playbook is designed and developed to deploy Docker Enterprise Edition on Cisco UCS servers for both the architectures (the first and second architectures) as called out in the earlier sections. The playbook not only deploys but also meets the prerequisites on OS configuration and Storage considerations. Following are the prerequisites to run the Ansible playbook:

- Playbook is recommended to be executed from a build server having an Ansible version 2.3.2 or greater if the YUM role need to be executed, else the playbook can be run on any of the cluster nodes itself and a separate build server is not needed. Ensure that the password-less SSH access from the node on which the playbook is run, to itself, and to the rest of the nodes is configured.
- The playbook is designed to accept some of the environment variables to be setup as group_vars, see the playbook tree structure below.
- This playbook is designed for the environments that works behind proxy, if proxy is not required certain **tasks defined in the ‘templates’** can be skipped.
- The environmental files like /etc/hosts, Cisco VIC enic driver RPM etc are predefined and are **configured ‘common’ role**; as the name suggests these are common to all the nodes.
- All required configuration files for DEE installation has been supplied through Ansible templates.
- Playbook does all the post OS installation tasks including storage configurations, firewall and Docker EE installation.
- Playbook is created with the UCS-Manager version 3.2(2b) and corresponding Cisco VIC - enic driver for RHEL7.3.

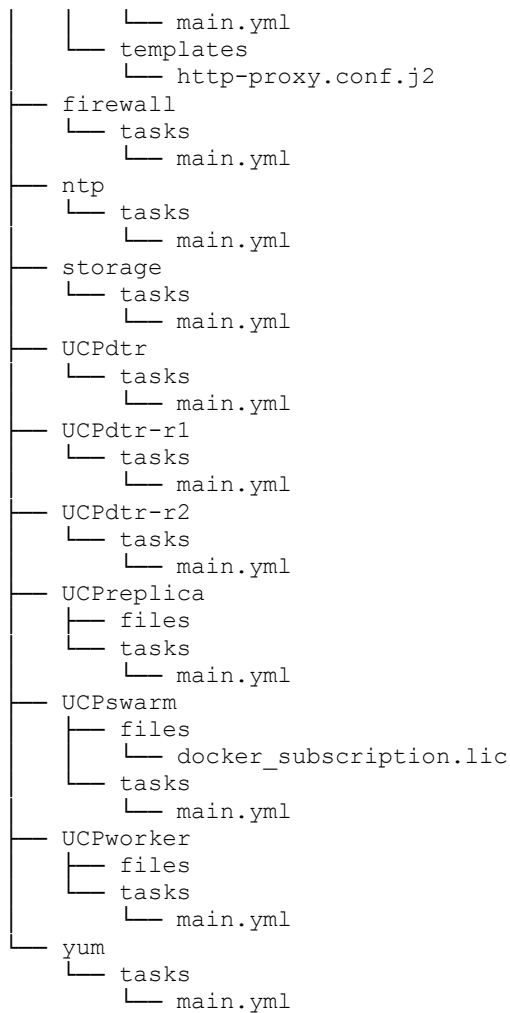


Ansible Playbook created for Docker EE install on Cisco UCS bare metal servers can be downloaded from: https://github.com/CiscoUcs/DockerEE_UCS_BM

Ansible Playbook Tree Structure

The following code tree structure shows the various roles defined in our Ansible playbook:

```
#
.
├── ansible.cfg
├── DEE-C-Nodes
├── DEE-C-Nodes.yml
├── DEE-Nodes
├── DEE-Nodes.yml
├── group_vars
│   └── all
├── hosts
├── roles
│   ├── common
│   │   ├── files
│   │   │   ├── kmod-enic-2.3.0.39-rhel7u3.e17.x86_64.rpm
│   │   │   └── kmod-enic-2.3.0.44-rhel7u3.e17.x86_64.rpm
│   │   ├── tasks
│   │   │   └── main.yml
│   │   └── templates
│   │       ├── bash_profile.j2
│   │       ├── environment.j2
│   │       ├── hosts.j2
│   │       ├── ntp.conf.j2
│   │       └── rhsm.conf.j2
│   └── docker
│       ├── files
│       │   └── daemon.json
│       └── tasks
```



33 directories, 29 files

Ansible Playbook Global Variables (group_vars)

Ansible group_vars/all is defined for all the roles and tasks, details of where it is applicable are shown in the following table:

Table 6 Ansible global variables (group_vars)

Variables	Values	Purpose and Usage
dee_url	<URL as given by Docker>	Used for downloading and installing Docker EE engine on cluster nodes <i>docker/tasks/main.yml: echo {{dee_url}} > /etc/yum/vars/dockerurl</i> <i>docker/tasks/main.yml: yum-config-manager --add-repo {{dee_url}}/docker-ee.repo</i>
ntp_server/ntp_se	<NTP server IP>	To configure NTPd services on the cluster nodes

server2		<pre>common/templates/ntp.conf.j2:server {{ ntp_server }} common/templates/ntp.conf.j2:server {{ ntp_server2 }} ntp/tasks/main.yml: ntpdate -q {{ntp_server}} ntp/tasks/main.yml: ntpdate {{ntp_server}} ntp/tasks/main.yml: ntpdate -q {{ntp_server}}</pre>
http_proxy/https_proxy/no_proxy/proxy_port	<URL/port>	<p>Used for proxy/environment settings for Docker Engine</p> <pre>common/templates/rhsm.conf.j2:proxy_hostname = {{ http_proxy_hostname }} common/templates/bash_profile.j2:export http_proxy={{ http_proxy }} common/templates/environment.j2:export http_proxy={{ http_proxy }} docker/templates/http-proxy.conf.j2:Environment="HTTP_PROXY={{ http_proxy }}" common/templates/rhsm.conf.j2:proxy_port = {{ proxy_port }}</pre>
node*/node*_fqdn	<node_name>/<node_fqdn>	<p>For creating a common /etc/hosts file across all the cluster nodes</p> <pre>common/templates/hosts.j2:{{ node01 }} {{ node01_fqdn }} {{ node01_name }}</pre>
rhsm_user/rhsm_password	<user name/password>	<p>User name/password for RHSM registration of the cluster nodes</p> <pre>yum/tasks/main.yml: subscription-manager register --username={{rhsm_user}} --password={{rhsm_password}}</pre>
pool_id	<subscription pool id for RHSM>	<p>Needed for repo attachments</p> <pre>yum/tasks/main.yml: subscription-manager attach --pool={{pool_id}}</pre>
UCP_Manager	<IP address of the first UCP manage host>	<p>First UCP manage node identification</p> <pre>UCPreplica/tasks/main.yml: docker swarm join --token {{ hostvars[groups['UCP-Mgr'][0]]['mgrtoken']['stdout'] }} {{ UCP_Manager }}:2377 UCPswarm/tasks/main.yml: --host-address {{UCP_Manager}} --controller-port {{UCP_Port}} --admin-username {{UCP_Admin}} --admin-password</pre>

		<pre> {{UCP_Admin_Pass}} \ UCPworker/tasks/main.yml: docker swarm join --token {{ hostvars[groups['UCP-Mgr'][0]]['wrktoken']['stdout'] }} {{ UCP_Manager }}:2377 </pre>
UCP_Admin/UCP_Admin_Pass	<user name/password>	<pre> UCP Admin name and password UCPdtr/tasks/main.yml: --nfs-storage-url {{DTR_NFS_URL}} --ucp-password {{UCP_Admin_Pass}} - -ucp-url {{UCP_URL}} --ucp-username {{UCP_Admin}} UCPdtr-r1/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- url {{UCP_URL}} \ UCPdtr-r1/tasks/main.yml: --ucp-username {{UCP_Admin}} --existing-replica-id {{hostvars[groups['UCP-DTR'][0]]['replicaid']['stdout']}} UCPdtr-r2/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- url {{UCP_URL}} \ UCPdtr-r2/tasks/main.yml: --ucp-username {{UCP_Admin}} --existing-replica-id {{hostvars[groups['UCP-DTR'][0]]['replicaid']['stdout']}} UCPswarm/tasks/main.yml: --host-address {{UCP_Manager}} --controller-port {{UCP_Port}} --admin- username {{UCP_Admin}} --admin-password {{UCP_Admin_Pass}} \ </pre>
DTR_NFS_URL	<NFS file share URL>	<pre> For configuring common NFS file system for DTR storage access UCPdtr/tasks/main.yml: --nfs-storage-url {{DTR_NFS_URL}} --ucp-password {{UCP_Admin_Pass}} - -ucp-url {{UCP_URL}} --ucp-username {{UCP_Admin}} </pre>
UCP_URL	<UCP URL value>	<pre> For DTR installation and integration with UCP Swarm cluster UCPdtr/tasks/main.yml: --nfs-storage-url {{DTR_NFS_URL}} --ucp-password {{UCP_Admin_Pass}} - -ucp-url {{UCP_URL}} --ucp-username {{UCP_Admin}} UCPdtr-r1/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- </pre>

		<pre>url {{UCP_URL}} \ UCPdtr-r2/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- url {{UCP_URL}} \</pre>
UCP_Port	<port>	<p>For UCP URL port config, for second architecture where UCP/DTR services are co-hosted, this needs to set to other than 443, as DTR by default uses 443 port</p> <pre>firewall/tasks/main.yml: firewall-cmd --zone=public -- add-port={{ UCP_Port }}/tcp firewall/tasks/main.yml: firewall-cmd --zone=public -- add-port={{ UCP_Port }}/tcp --permanent UCPswarm/tasks/main.yml: --host-address {{UCP_Manager}} --controller-port {{UCP_Port}} --admin- username {{UCP_Admin}} --admin-password {{UCP_Admin_Pass}} \</pre>
UCP_Ver/DTR_Ver	<version values>	<p>For fixing UCP/DTR version values</p> <pre>UCPswarm/tasks/main.yml: docker container run --rm -t --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp:{{UCP_Ver}} install \ UCPdtr/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} install --ucp-node `hostname` -- ucp-insecure-tls \ UCPdtr-r1/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- url {{UCP_URL}} \ UCPdtr-r2/tasks/main.yml: docker run -t --rm docker/dtr:{{DTR_Ver}} join --ucp-node `hostname` --ucp- insecure-tls --ucp-password {{UCP_Admin_Pass}} --ucp- url {{UCP_URL}} \</pre>

Ansible Playbook Roles

This file contains a list of roles that gets configured on the Docker EE cluster nodes. They include following tasks:

1. Common – This role takes care of all common tasks across the cluster nodes which include setting up environment values, proxy settings, files which are common to all nodes. Users need to tweak these values inside roles/common/templates and roles/common/files to suit their environment.



hosts.j2 file is a template and has cluster node details, which gets populated on cluster nodes as /etc/hosts.

2. Yum – This role includes tasks for registering cluster nodes to RHSM network, attach to pool_id, repos, update enic driver and do `yum update`. This task also takes care for host rebooting post yum update and handing over control over to following roles/tasks execution.
3. Ntp – For configuring NTPd services on cluster nodes. NTP services are critical for DEE services functioning.
4. Firewall – This role takes care of setting hosts firewall ports for Docker Enterprise Edition infra-services requirements.
5. Storage – **This solution leverages UCSM's storage profile policy to configure blades and rack server's** local storage disks for Docker Enterprise Edition. This gives RAID-1 and RAID-10 configuration for redundancies and performance out of storage sub-system. 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. This Ansible role is responsible for storage configuration.
6. Docker – This role takes care of installing Docker EE engine on all the cluster nodes with required version and dependencies.
7. UCPswarm – This role is for initializing Swarm on first node of the cluster and installing/configuring UCP services. This also includes attaching Docker EE license file to be configured to the UCP install. File can be copied to `files` under the role/UCPswarm.
8. UCPreplica – This role joins other manager nodes into Swarm and makes a cluster of managers. It also clusters UCP services.
9. UCPworker – Role for joining rest of the nodes into Swarm cluster as worker nodes.
10. UCP-DTR – For installing DTR stack on designated UCP worker nodes. It also takes care of configuring common NFS share file system for image repository.
11. UCP-DTR-R1/R2 – Roles for DTR replicas on worker nodes.

Sizing Considerations

For the production grade Docker Enterprise Edition deployment Docker recommends infrastructure services to be run on dedicated hosts. There are two infrastructure components, namely – Docker UCP Managers and DTR. Container workloads are to run on dedicated node termed as Docker UCP Workers. For production requirements, one each of Docker UCP Manager and DTR service node should be running in cluster of bare metal servers. In order to sustain a minimum of one node failure, it is recommended to have three-nodes for running both UCP Manager and DTR services separately. Based on this recommendation, Docker UCP Manager and DTR services will run on three node clusters separately within the swarm cluster. This solution proposes a ten node setup, where six nodes are consumed by the Docker **EE's infrastructure (UCP Manager and DTR)** services and the remaining four nodes take up the Container workload. Administrators have an option to configure the UCP Manager 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 the same three-node cluster. With this design, the overall Docker Enterprise

Edition deployment gets optimized to a four node cluster configuration. The fourth node runs as a UCP Worker node to take the container workload. The scheduler configuration settings allow administrators and users to deploy containers on the UCP Manger **nodes and using the scheduling strategy set to 'spread'**, all the nodes will be available for container deployment.

Maximum number of containers deployed per node depends on the type of containers and the applications that are run within the containers. This solution is designed and validated to spin 300 containers on each pf the UCP Nodes.

Adding another node to the cluster follows a simple procedure with a minimal manual intervention. With a policy based logical server model, achieved through Cisco UCS Manager, scaling-up the bare metal 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 an additional hardware available in the server pool. Node addition workflow is covered later in the Scale Tests section.

Software and Hardware Versions

The following tables provide software and hardware versions used in this solution for both the architectures. Following table lists the Cisco UCS infrastructure components used in the solution for production deployment.

Table 7 Solution Component Details for the first architecture

Component	Model	Quantity	Comments
UCP mangers, UCP Worker and UCP Worker hosting DTR services	Cisco UCS B200 M5 Servers	10	CPU - 2 x Intel Xeon Gold E7 6130@2.1GHz Memory - 12 x 16GB@2666 MHz RDIMM DIMMs - total of 192GB Local Disks - 2 x 300 GB SAS disks for OS Boot and Docker Engine Network Card - 1x1340 VIC Raid Controller - Cisco MRAID 12 G SAS Controller
Chassis	Cisco UCS 5108 Chassis	2	
IO Modules	Cisco UCS 2304XP Fabric Extenders	4	
Fabric Interconnects	Cisco UCS 6332-16-UP Fabric Interconnects	2	
TOR Switches	Cisco Nexus 9396PX Switches	2	

List of hardware platform and software versions used in the first architecture.

Table 8 Hardware and Software versions for the first architecture

Layer	Device	Image	Comments
Computing	Cisco UCS B200 M5 Servers	Version 3.2 (2b)	Cisco UCS server
Network Adapter	Cisco UCS 1340 Virtual Interface Card (VIC)	Version 3.2 (2b)	Cisco VIC firmware
Network	Cisco UCS 6332-16UP Fabric Interconnects	Version 3.2 (2b)	Cisco UCS Fabric Interconnect
	Cisco Nexus 9396PX Switches	Version 7.0(3)I4(7)	Cisco Nexus TOR Switch
Cisco Software	Cisco UCS Manager	Version 3.2 (2b)	Cisco UCS Manager
Contiv	Contiv Netplugin	1.1.7	Contiv v2plugin for DEE
Docker Enterprise Edition	Docker EE Engine	Version 17.06.2-ee-5	Docker Enterprise Edition Engine Note: This solution has been validated on this version of Docker Engine.
	Docker Swarm	Version 2.2.4	Docker Swarm Scheduler is embedded in UCP Note: Docker Swarm version is appropriately picked during the UCP install.
	Docker Universal Control Plane (UCP)	Version 2.2.4	Docker Environment Orchestrator and Management Interface Note: This solution has been validated on this version of Docker UCP.
	Docker Trusted Repository (DTR)	Version 2.4.0	Docker Image Store for Enterprise Note: This solution has been validated on this version of DTR
Operating System (OS)	Red Hat Enterprise Linux	Version 7.3	Red Hat Linux for bare metal OS
NIC Driver	Cisco UCS 1340 Virtual Interface Card (VIC)	Version 2.3.0.44	Cisco eNIC device driver for RHEL 7.3 OS

Below 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 Manager, UCP and DTR worker nodes	Cisco UCS C220 M5 Servers	4	CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB Local Disks – 8 x 600 GB SAS disks for OS Boot and Docker Engine Network Card – 1x1385 VIC Raid Controller – Cisco MRAID 12 G SAS Controller
Fabric Interconnects	Cisco UCS 6332-16UP Fabric Interconnects	2	
TOR Switches	Cisco Nexus 9396PX Switches	2	

List of hardware platform and software versions used in the second architecture.

Table 10 Hardware and Software Versions for the second architecture

Layer	Device	Image	Comments
Computing	Cisco UCS C220 M5 Servers	Version 3.2 (2b)	Cisco UCS server
Network Adapter	Cisco UCS 1385 Virtual Interface Card (VIC)	Version 3.2 (2b)	Cisco VIC firmware
Network	Cisco UCS 6332-16UP Fabric Interconnects	Version 3.2 (2b)	Cisco UCS Fabric Interconnect
	Cisco Nexus 9396PX Switches	Version 7.0(3)I4(7)	Cisco N9K TOR Switch
Cisco Software	Cisco UCS Manager	Version 3.2 (2b)	Cisco UCS Manager
Contiv	Contiv Netplugin	1.1.7	Contiv v2plugin for DEE
Docker Enterprise Edition	Docker EE Engine	Version 17.06.2-ee-5	Docker Enterprise Edition Engine Note: This solution has been validated on this version of Docker Engine.
	Docker Swarm	Version 2.2.4	Docker Swarm Scheduler is embedded in UCP Note: Docker Swarm version is appropriately picked during the UCP

			install.
	Docker Universal Control Plane (UCP)	Version 2.2.4	Docker Environment Orchestrator and Management Interface Note: This solution has been validated on this version of Docker UCP.
	Docker Trusted Repository (DTR)	Version 2.4.0	Docker Image Store for Enterprise Note: This solution has been validated on this version of DTR
Operating System (OS)	Red Hat Enterprise Linux	Version 7.3	Red Hat Linux for bare metal OS
NIC Driver	Cisco UCS 1340 Virtual Interface Card (VIC)	Version 2.3.0.44	Cisco eNIC device driver for RHEL 7.3 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]:

```

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 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:

1. On each Nexus 9000, enter the configuration mode:
`config terminal`
2. Use the following commands to enable the necessary features:
`feature udd`
`feature lacp`
`feature vpc`
`feature interface-vlan`
3. 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 - Administration

Initial Setup of Cisco Fabric Interconnects

A pair of Cisco UCS 6332-16UP 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 6332-16UP 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 6332-16UP FI - Secondary (FI-B)

1. Connect to the console port on the second FI on Cisco UCS 6332-16UP 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 6332-16UP Fabric Interconnect cluster IP address configured in earlier step.
3. Click Launch Cisco UCS Manager link to download the Cisco UCS Manager software.
4. If prompted, accept security certificates as necessary.
5. When prompted, enter admin as the user name and enter the administrative password.
6. Click Login to log in to Cisco UCS Manager.
7. 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](#).

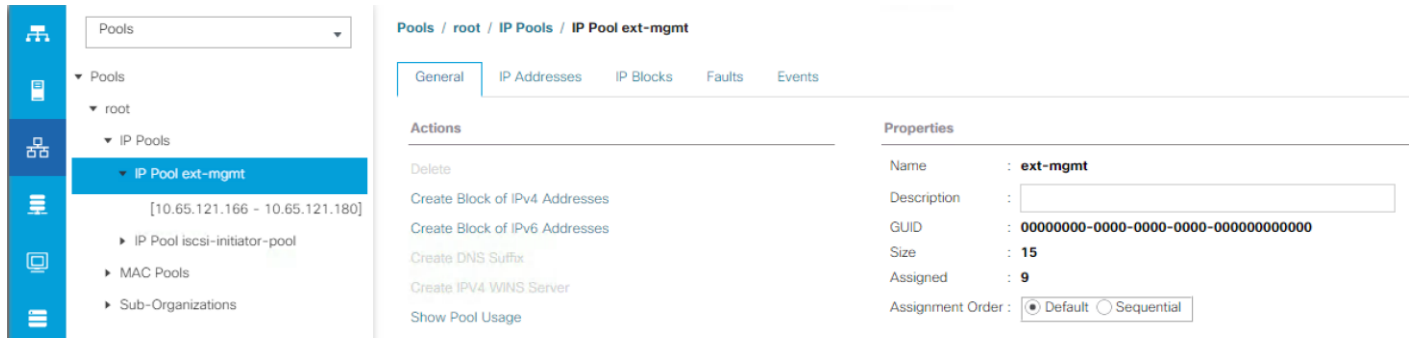
Assigning Block of IP addresses for KVM Access

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. And should be configured for out-of-band access.

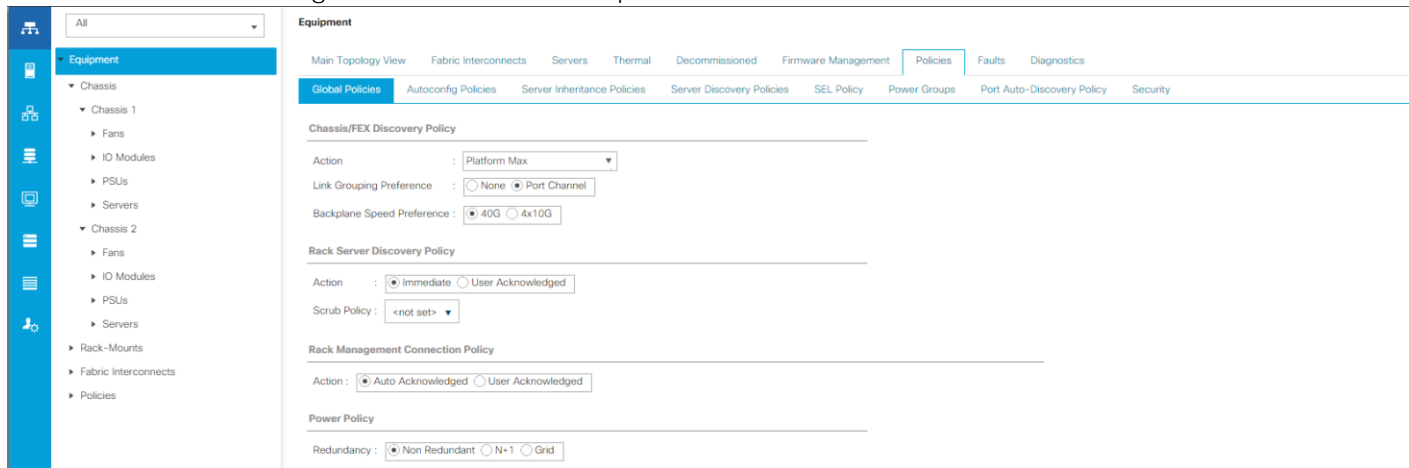
1. From Cisco UCS Manager, click LAN tab in the navigation pane.
2. Select LAN > Pools > root > IP Pools.
3. Right-click and select Create IP Pool.
4. Specify a Name (for example, ext-mgmt) for the pool. Click Next.
5. Click [+] Add to add a new IP Block. Click Next.
6. Enter the starting IP address (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.

Fabric Interconnects / Fabric Interconnect A (primary) / Fixed Module / Ethernet Ports

Slot	Aggr. Port ID	Port ID	MAC	If Role	If Type	Overall Status	Admin State
1	0	7	8C:60:4F:BD:2F:DA	Server	Physical	Up	Enabled
1	0	8	8C:60:4F:BD:2F:DB	Server	Physical	Up	Enabled
1	0	9	8C:60:4F:BD:2F:DC	Server	Physical	Up	Enabled
1	0	10	8C:60:4F:BD:2F:DD	Server	Physical	Up	Enabled
1	0	11	8C:60:4F:BD:2F:DE	Server	Physical	Up	Enabled
1	0	12	8C:60:4F:BD:2F:DF	Server	Physical	Up	Enabled
1	0	13	8C:60:4F:BD:2F:E0	Server	Physical	Up	Enabled
1	0	14	8C:60:4F:BD:2F:E1	Server	Physical	Up	Enabled
1	0	15	8C:60:4F:BD:2F:E2	Server	Physical	Up	Enabled

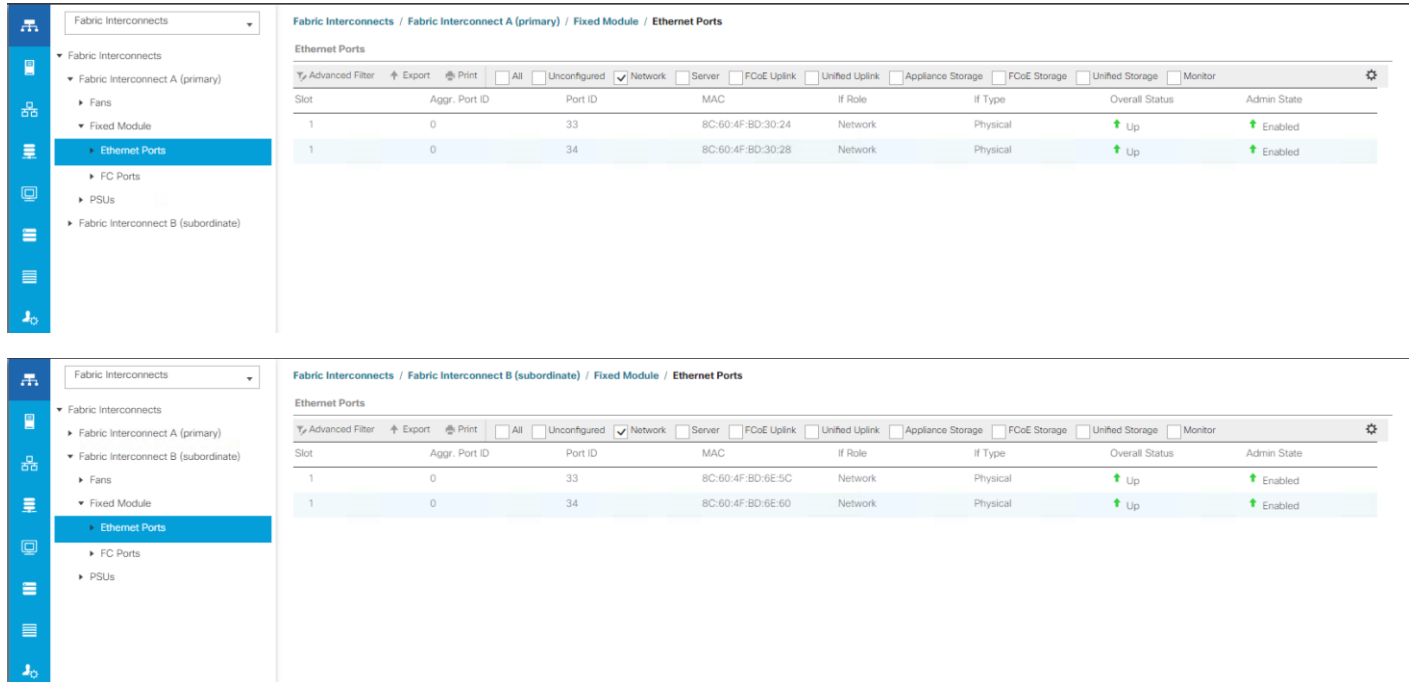
Fabric Interconnects / Fabric Interconnect B (subordinate) / Fixed Module / Ethernet Ports

Slot	Aggr. Port ID	Port ID	MAC	If Role	If Type	Overall Status	Admin State
1	0	7	8C:60:4F:BD:6E:12	Server	Physical	Up	Enabled
1	0	8	8C:60:4F:BD:6E:13	Server	Physical	Up	Enabled
1	0	9	8C:60:4F:BD:6E:14	Server	Physical	Up	Enabled
1	0	10	8C:60:4F:BD:6E:15	Server	Physical	Up	Enabled
1	0	11	8C:60:4F:BD:6E:16	Server	Physical	Up	Enabled
1	0	12	8C:60:4F:BD:6E:17	Server	Physical	Up	Enabled
1	0	13	8C:60:4F:BD:6E:18	Server	Physical	Up	Enabled
1	0	14	8C:60:4F:BD:6E:19	Server	Physical	Up	Enabled
1	0	15	8C:60:4F:BD:6E:1A	Server	Physical	Up	Enabled

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

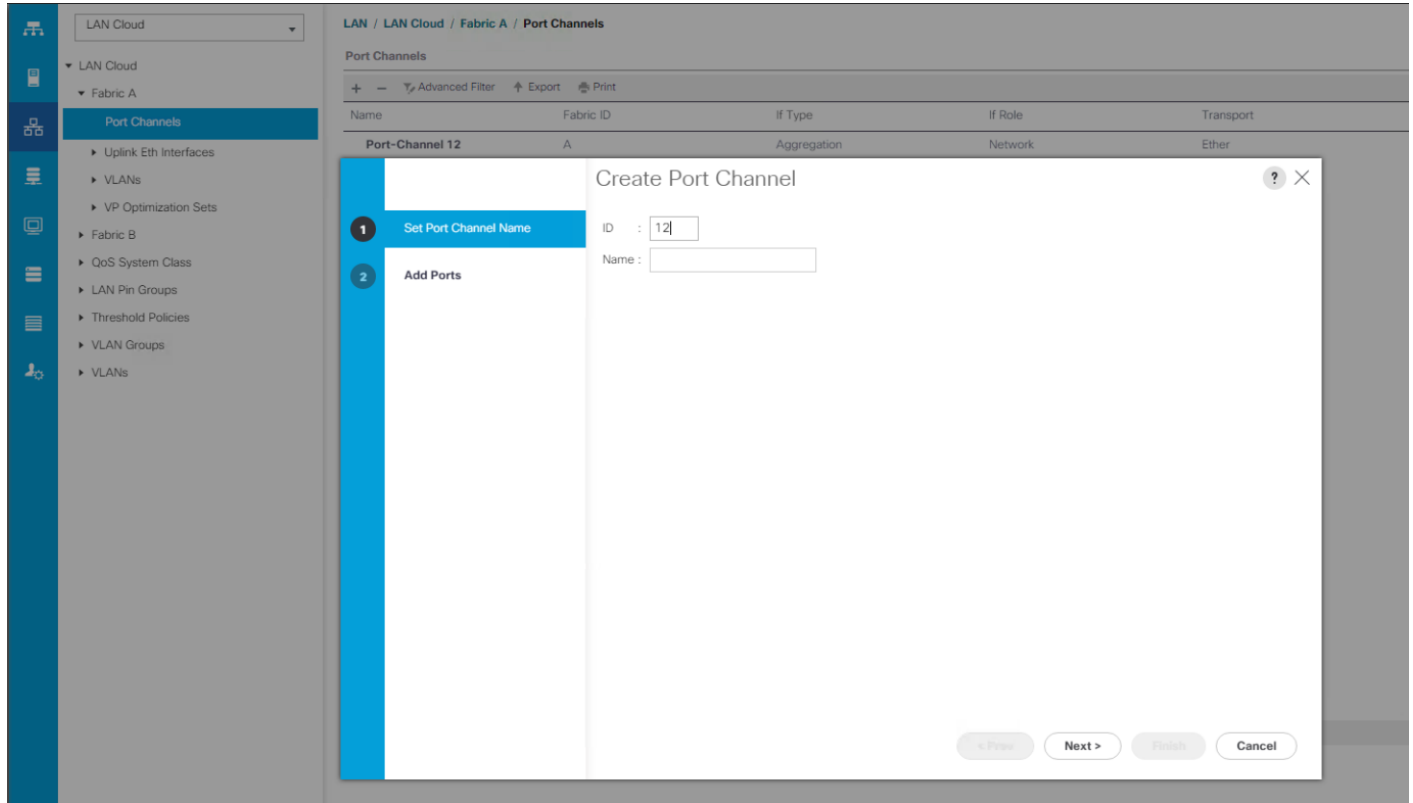


Configuring Port Channels on Uplink Ports to Cisco Nexus 9000 Series Switches

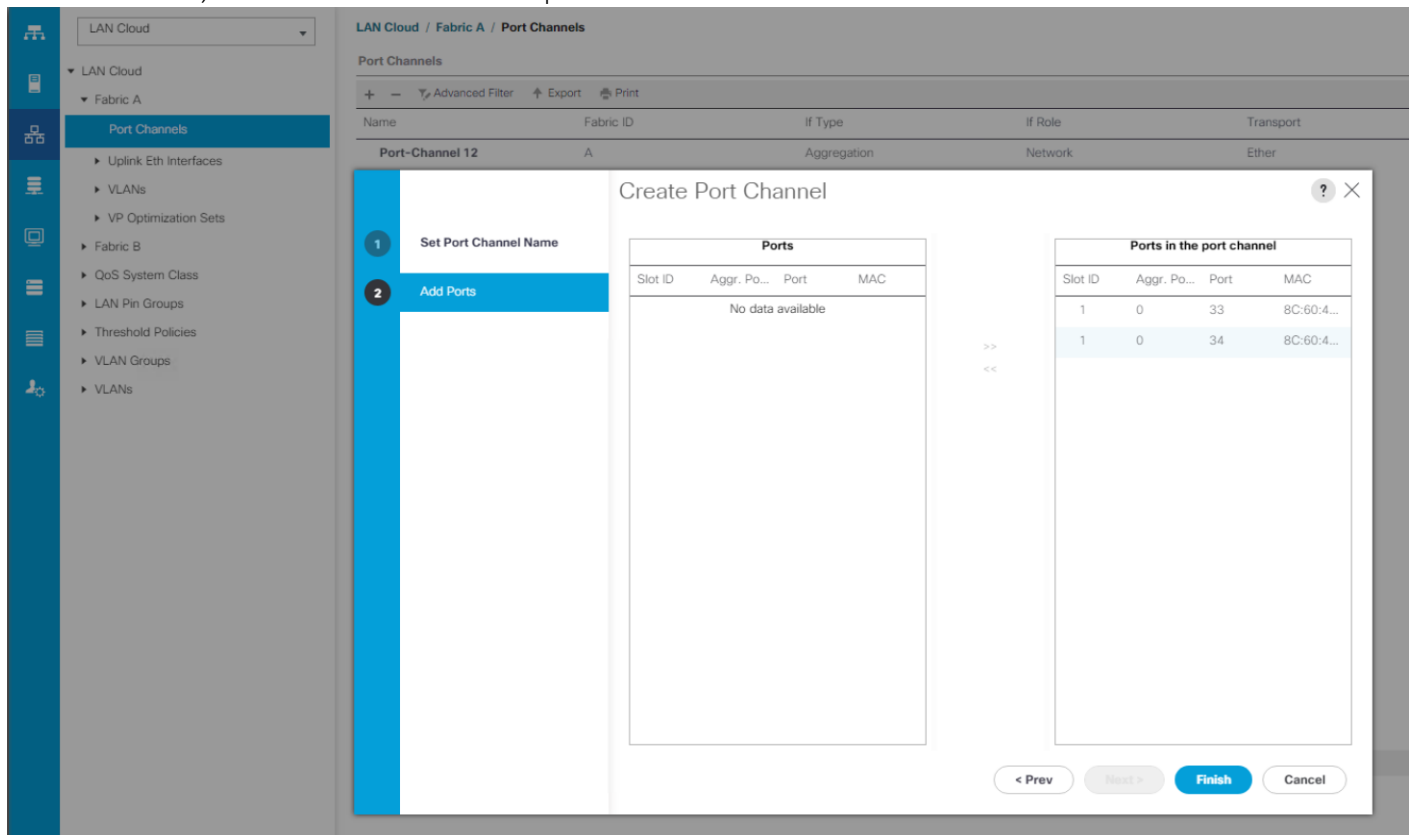
In this procedure, two port channels are created, one from Fabric A to both the Cisco Nexus switches and one from Fabric B to both the Cisco Nexus switches.

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, Eth1/33 and Eth1/34). Click Finish to create the port channel.



6. Verify the resulting configuration.

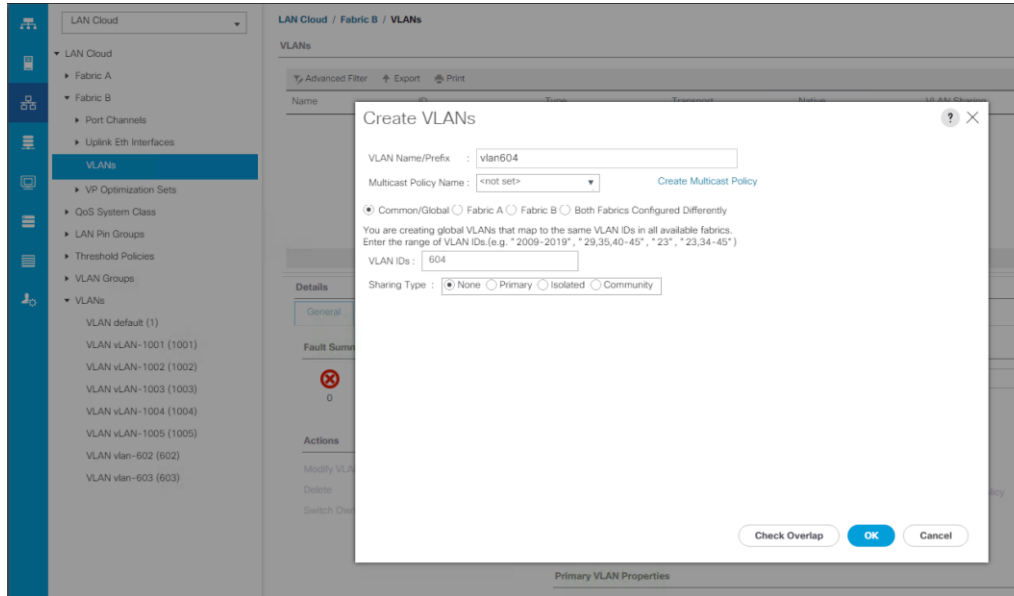
7. Repeat above steps for Fabric B and verify the configuration.

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, vln604) and VLAN ID (for example, 604).



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. As this solution uses Contiv Network plugin in L2 VLAN mode, pre-provisioning of user VLANs is important. There are 2 sets of VLAN needed:

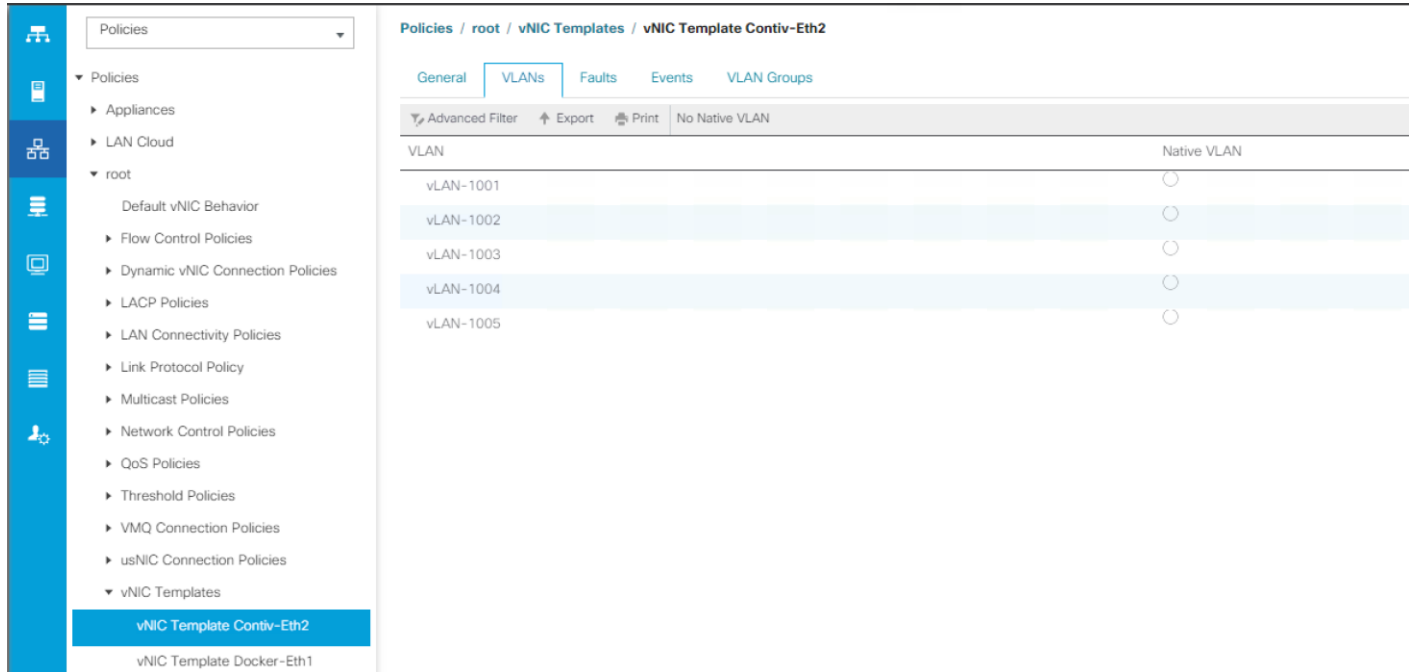
- Externally accessible VLAN for DEE/Contiv hosts in the cluster
- VLAN range to be used for Contiv networking backplane to be consumed by container work-load

Host access VLAN will be used on vNIC1 while Contiv VLANs will be consumed on vNIC2. Users can define the range of VLAN to be used based on their need.



Unless VLANs are configured on appropriate vNIC, to be used for Contiv data-path, data forwarding will not happen. All the Contiv VLANs configured should always be non-native.

An example of Contiv VLANs configured for Container data-path on vNIC2 is given below:

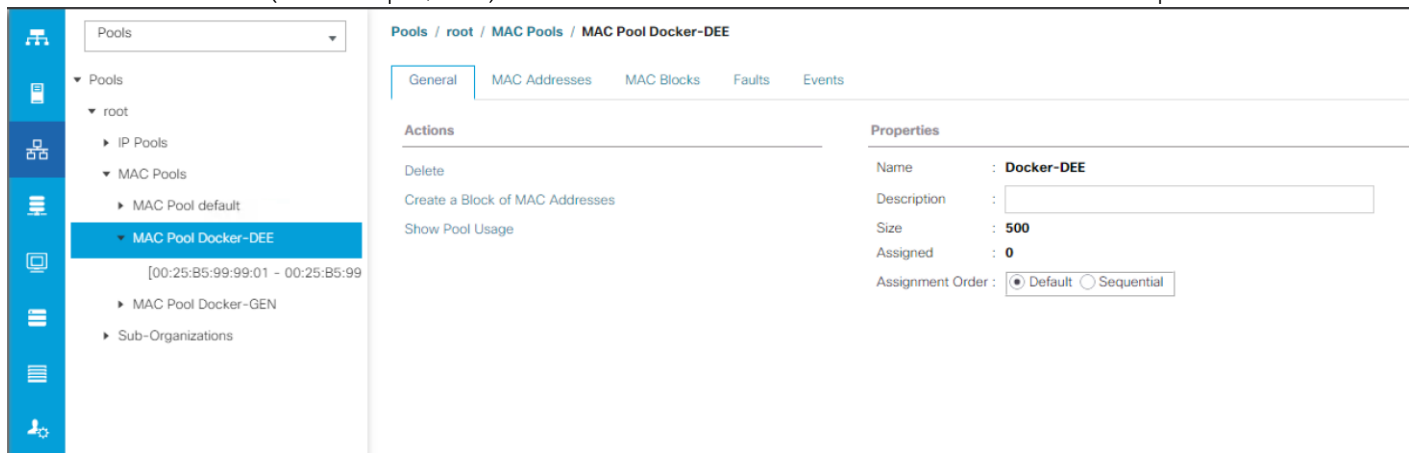


Creating LAN Pools

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-EE) 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 99:99 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, 500) 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 B – Contiv Data-path

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.

Create vNIC Template ?

Name :

Description :

Fabric ID : Fabric A Fabric B Enable Failover

Redundancy

Redundancy Type : No Redundancy Primary Template Secondary Template

Target

Adapter VM

Warning

If **VM** is selected, a port profile by the same name will be created.
 If a port profile of the same name exists, and updating template is selected, it will be overwritten

Template Type : Initial Template Updating Template

VLANs VLAN Groups

Advanced Filter Export

Select	Name	Native VLAN
<input type="checkbox"/>	default	<input type="radio"/>
<input checked="" type="checkbox"/>	vLAN-1001	<input type="radio"/>
<input checked="" type="checkbox"/>	vLAN-1002	<input type="radio"/>
<input checked="" type="checkbox"/>	vLAN-1003	<input type="radio"/>
<input checked="" type="checkbox"/>	vLAN-1004	<input type="radio"/>
<input checked="" type="checkbox"/>	vLAN-1005	<input type="radio"/>

Create VLAN

CDN Source : vNIC Name User Defined

CDN Name :

MTU :

MAC Pool : ▼

QoS Policy : ▼

Network Control Policy : ▼

Pin Group : ▼

Stats Threshold Policy : ▼

Connection Policies

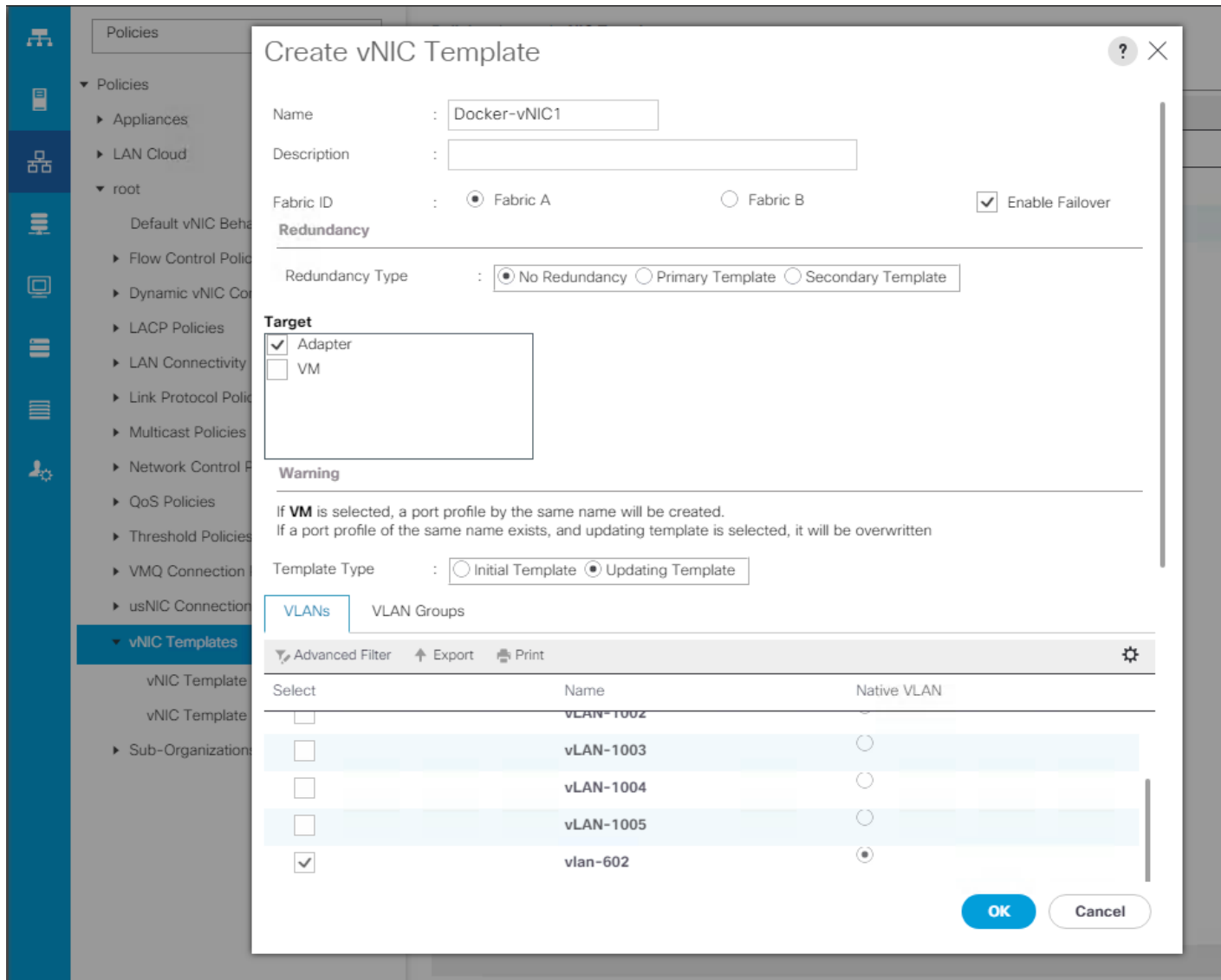
Dynamic vNIC usNIC VMQ

Dynamic vNIC Connection Policy : ▼

14. Click OK to create the vNIC template.

Creating vNIC Template for Fabric A - Host Access Path

Repeat the above steps to create a vNIC template (for example, Docker-vNIC1) through Fabric A.



Cisco UCS Configuration – Server

Creating Server Policies

In this section creation of various server policies that are used in this solution are shown.

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) and change the Consistent Device Naming to enabled state. Click Next.

The screenshot shows the Cisco UCS Manager interface. On the left, the navigation pane is open to 'Policies / root / BIOS Policies / Docker'. The main content area shows the configuration for the 'Docker' BIOS Policy. The 'Name' is set to 'Docker' and the 'Owner' is 'Local'. The 'Reboot on BIOS Settings Change' checkbox is checked. Below the properties, there is a table for BIOS Settings:

BIOS Setting	Value
CDN Control	Enabled
Front panel lockout	Platform Default
POST error pause	Platform Default
Quiet Boot	Platform Default
Resume on AC power loss	Platform Default

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

This solution uses scriptable vMedia feature for UCS Manager to install bare metal Oses on DEE cluster nodes. This option is fully automated and does not require PXE boot and any other manual intervention. vMedia policy and boot policy are two major configuration items in order to get it working.

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, DEE-vMedia).
5. Boot mode should be set to Legacy and rest of the options should be left at default.
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 the boot lun name (for example, Boot-LUN). Click OK to add the local lun image. This device will be the target device for bare metal OS install and will be used for host boot-up.



LUN name here should match with the LUN name defined in the storage profile to be used in service profile templates.

7. Add CIMC mounted CD/DVD under CIMC Mounted vMedia section, next to the local LUN definition. This device will be mounted for accessing boot images required during installation.
8. Add again CIMC mounted HDD from the section device class which is CIMC Mounted vMedia. This device will be mounted for bare metal operating system installation configuration.
9. After the creation Boot Policy, you can view the created boot options as shown.

Servers / Policies / root / Boot Policies / Boot Policy DEE-vMedia

General Events

Delete: Name: DEE-vMedia

Show Policy Usage: Description:

Use Global: Owner: Local

Reboot on Boot Order Change:

Enforce vNIC/vHBA/iSCSI Name:

Boot Mode: Legacy Uefi

Warning

The type (primary/secondary) does not indicate a boot order presence.
The effective order of boot devices within the same device class (LAN/Storage/iSCSI) is determined by PCIe bus scan order.
If **Enforce vNIC/vHBA/iSCSI Name** is selected and the vNIC/vHBA/iSCSI does not exist, a config error will be reported.
If it is not selected, the vNIC/vHBAs are selected if they exist, otherwise the vNIC/vHBA with the lowest PCIe bus scan order is used.

Local Devices

Add Local Disk

- Add Local LUN
- Add Local JBOD
- Add SD Card
- Add Internal USB
- Add External USB
- Add Embedded Local LUN
- Add Embedded Local Disk

Add CD/DVD

- Add Local CD/DVD
- Add Remote CD/DVD

Add Floppy

- Add Local Floppy
- Add Remote Floppy

Add Remote Virtual Drive

Add vMedia

CIMC Mounted vMedia

- Add CIMC Mounted CD/DVD
- Add CIMC Mounted HDD

Boot Order

Name	Order	vNIC/vHBA/iSCSI...	Type	WWN	LUN Name	Slot Number	Boot P
Local LUN	1						
CIMC Mounted CD/DVD	2						
CIMC Mounted HDD	3						

Move Up Move Down Delete

See Local Boot Parameters

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, 3.2.2b).
5. Leave Simple selected.
6. Select the package versions for the different type of servers (Blade, Rack) in the deployment (for example, 3.2(2b) for Blade and Rack servers).
7. Click OK twice to create the host firmware package.

Policies / root / Host Firmware Packages / 3.2.2b

Host Firmware Packages

Select	Vendor	Model	PID	Presence	Version
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS M72KR-E	N20-AE0102	present	10.0.803.19
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS M72KR-Q	N20-AQ0102	present	03.09.17
<input checked="" type="checkbox"/>	Intel Corp.	Intel 10GbE Adapter	N2XX-AIPCI01	present	800006A4-1.810.8
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1280	UCSB-VIC-M82-8P	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1240	UCSB-MLOM-40G-01	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1340	UCSB-MLOM-40G-02	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1340	UCSB-MLOM-40G-03	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1380	UCSB-VIC-M83-8P	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS MLOM 1227T	UCSC-MLOM-C10T-02	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS VIC 1387	UCSC-MLOM-C40Q-03	present	4.2(2a)
<input checked="" type="checkbox"/>	Cisco Systems Inc	Cisco UCS MLOM 1227	UCSC-MLOM-CSC-02	present	4.2(2a)
<input checked="" type="checkbox"/>	Broadcom Corp.	Broadcom 57810	UCSC-PCIE-B3SFP	present	01.02.19
<input checked="" type="checkbox"/>	Emulex	Emulex 32 Gb Dual Port FC HBA	UCSC-PCIE-BD32GF	present	11.2.156.27

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, 100) that is sufficient to support the available blade or server resources.
7. Click OK, click Finish and click OK again to create UUID Pool.

Pools / root / UUID Suffix Pools / Pool Docker-DEE

Properties

Name : **Docker-DEE**

Description :

Prefix :

Size : **100**

Assigned : **0**

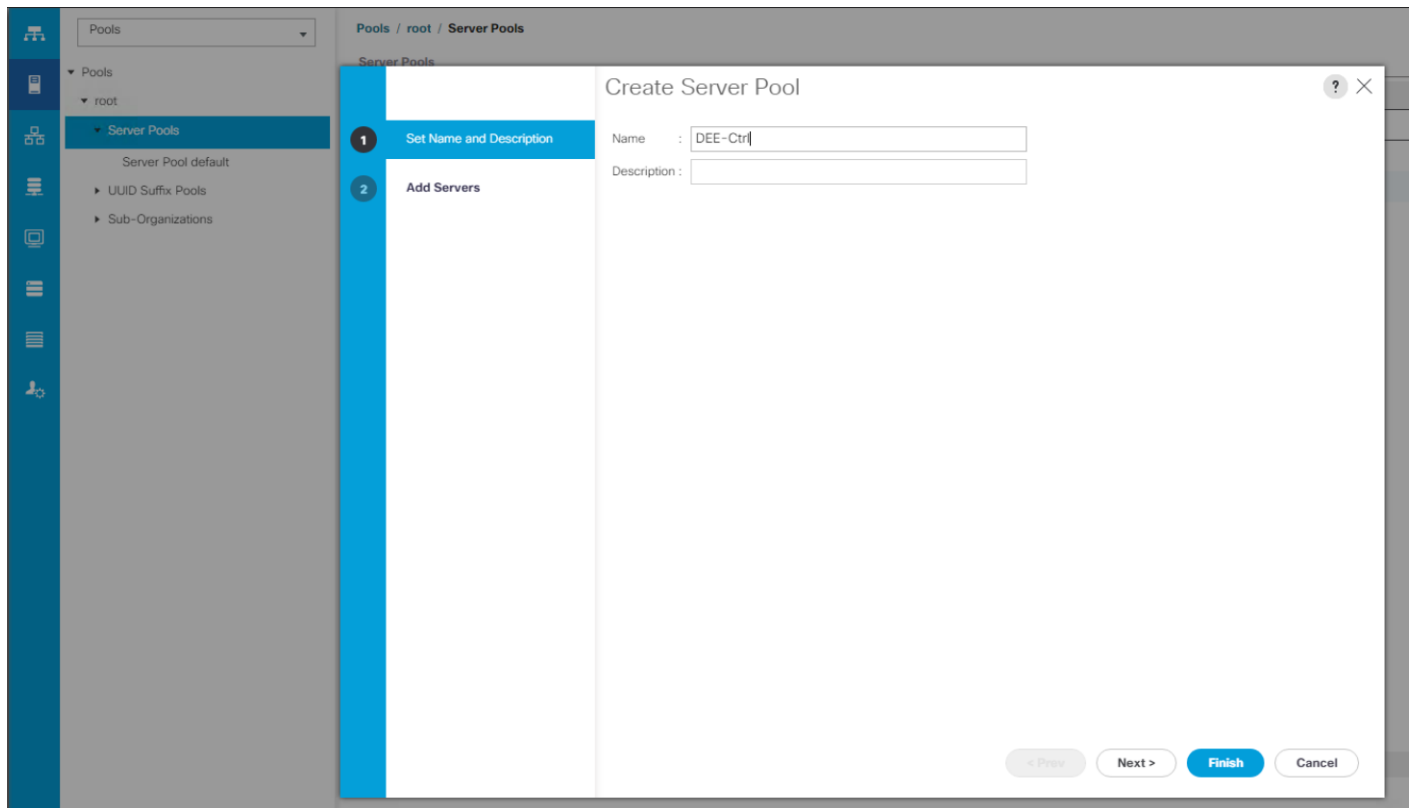
Assignment Order : Default Sequential

Creating Server Pools

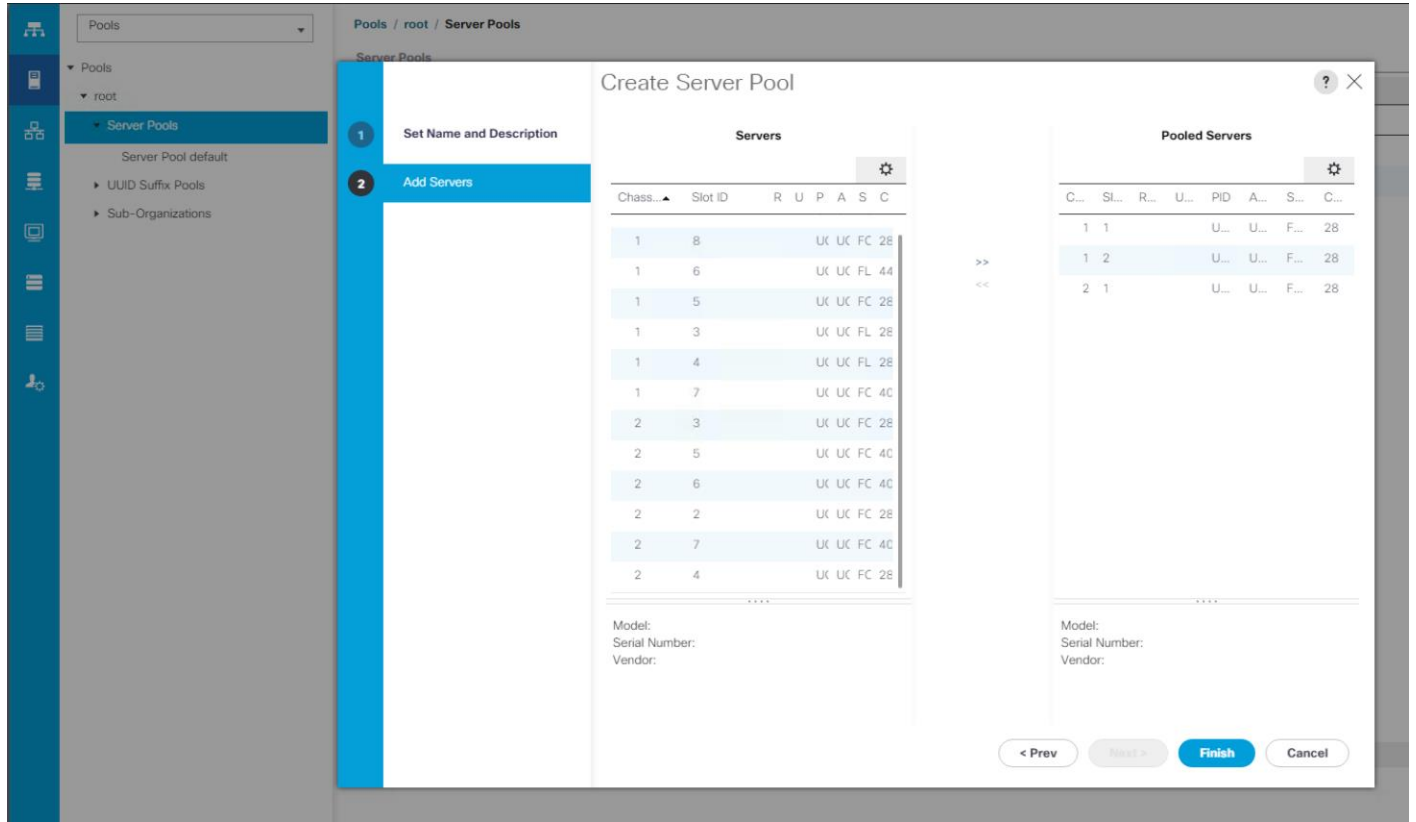
Three server pools are created, one each for DTR nodes, UCP master/controller nodes, and UCP worker nodes. Since there are three DTR nodes, three UCP-Master nodes and 4 UCP-Worker nodes in this solution, separate pools are created for each of these categories coming from two different chassis. This enables you to expand the cluster based on your requirements. Any of these categories of nodes can be scaled up, adding blade/s to the respective pool and creating new service profile from the corresponding template to add the new node to the DEE/Contiv cluster.

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

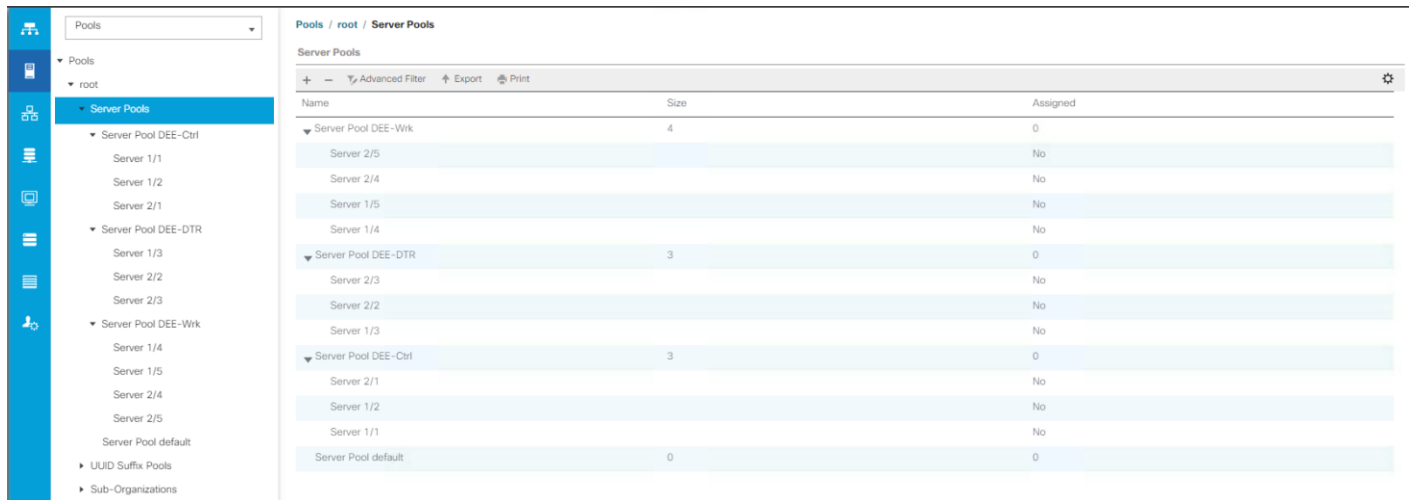
1. In Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Pools > root.
3. Right-click Server Pools and select Create Server Pool.
4. Enter name of the server pool (for example, DEE-Ctrl).
5. Optional: Enter a description for the server pool.
6. Click Next.



7. Select two (or more) servers to be added and click >> to add them to the server pool.



8. Click Finish to complete.
9. Similarly create two more Server Pools (for example, DEE-DTR and DEE-Wrk). 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:

Pools / root / Server Pools / Server Pool Docker

General Servers Faults Events

Name	Chassis ID	Slot ID	Assigned	Assigned To	Rack ID	Reason	Instance ID
Rack-Mount Server 1			Yes	org-root/ls-DEE-Wrk-C-1	1	Manually Added	
Rack-Mount Server 2			Yes	org-root/ls-DEE-Ctrl-C-3	2	Manually Added	
Rack-Mount Server 3			Yes	org-root/ls-DEE-Ctrl-C-2	3	Manually Added	
Rack-Mount Server 4			Yes	org-root/ls-DEE-Ctrl-C-1	4	Manually Added	

Cisco UCS Configuration – Storage

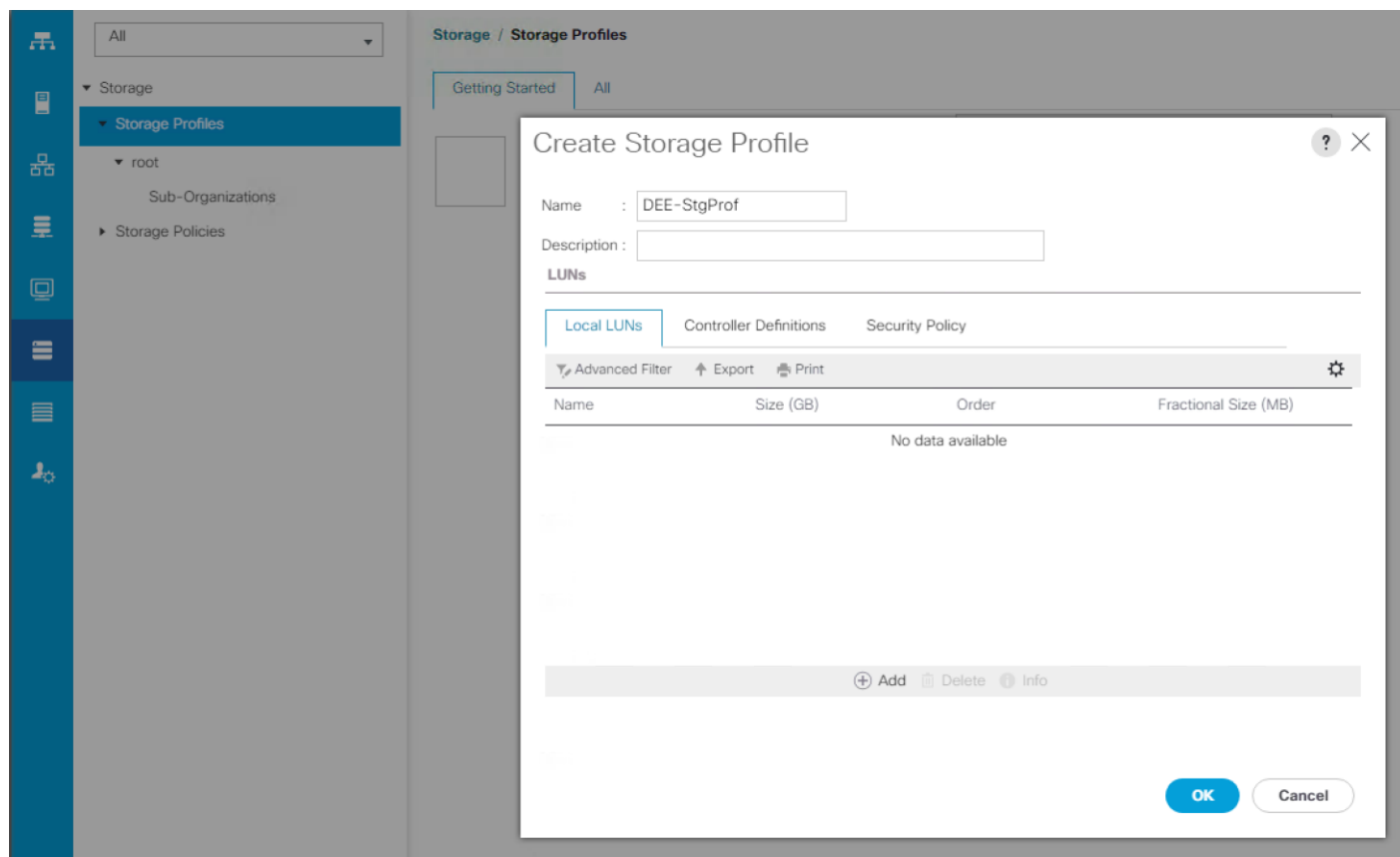
Creating Storage Profile

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.

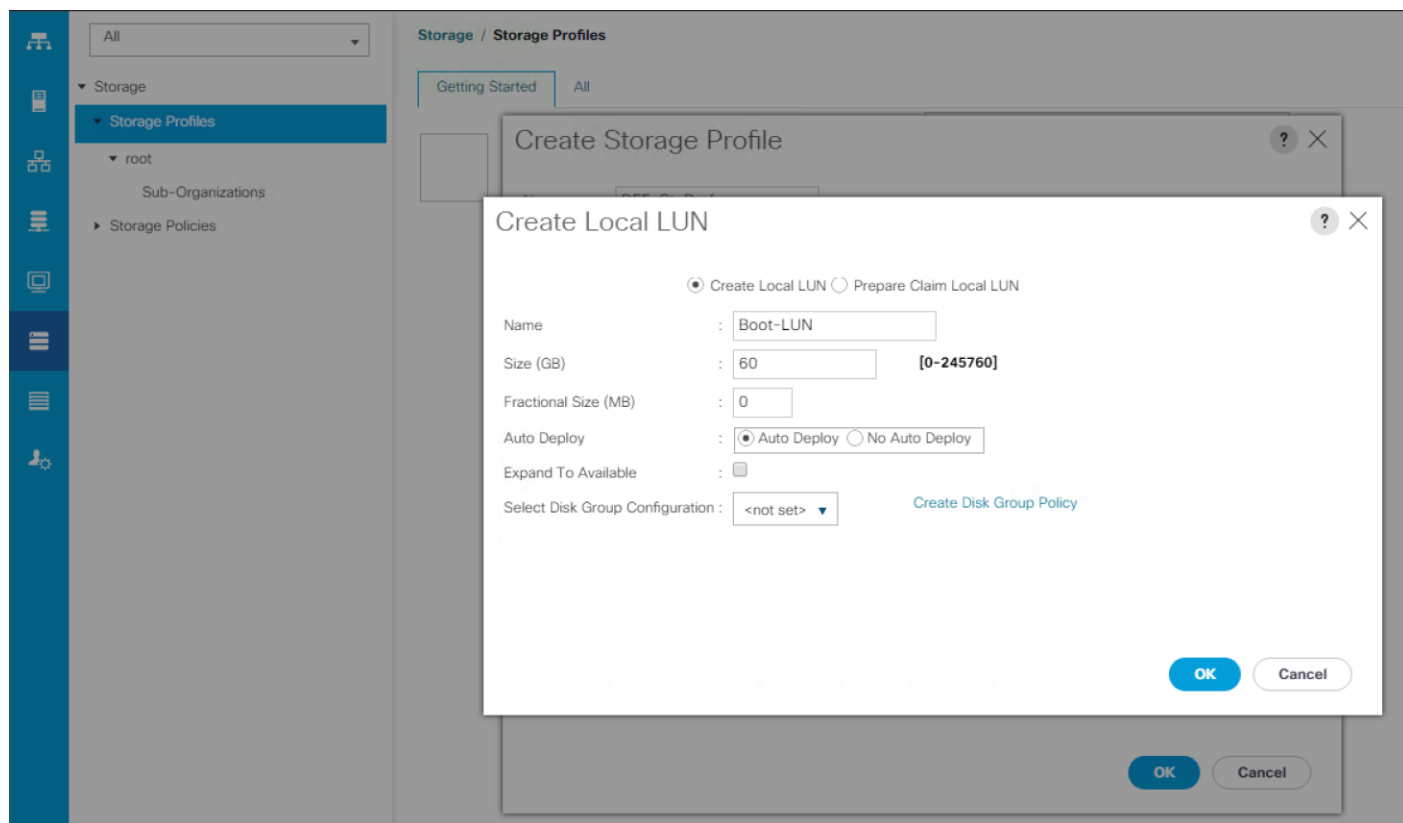
Having a storage profile created will reduce the task of configuring two virtual disks in the RAID Controller Option ROM or create a custom file system layout at the time of OS installation.

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

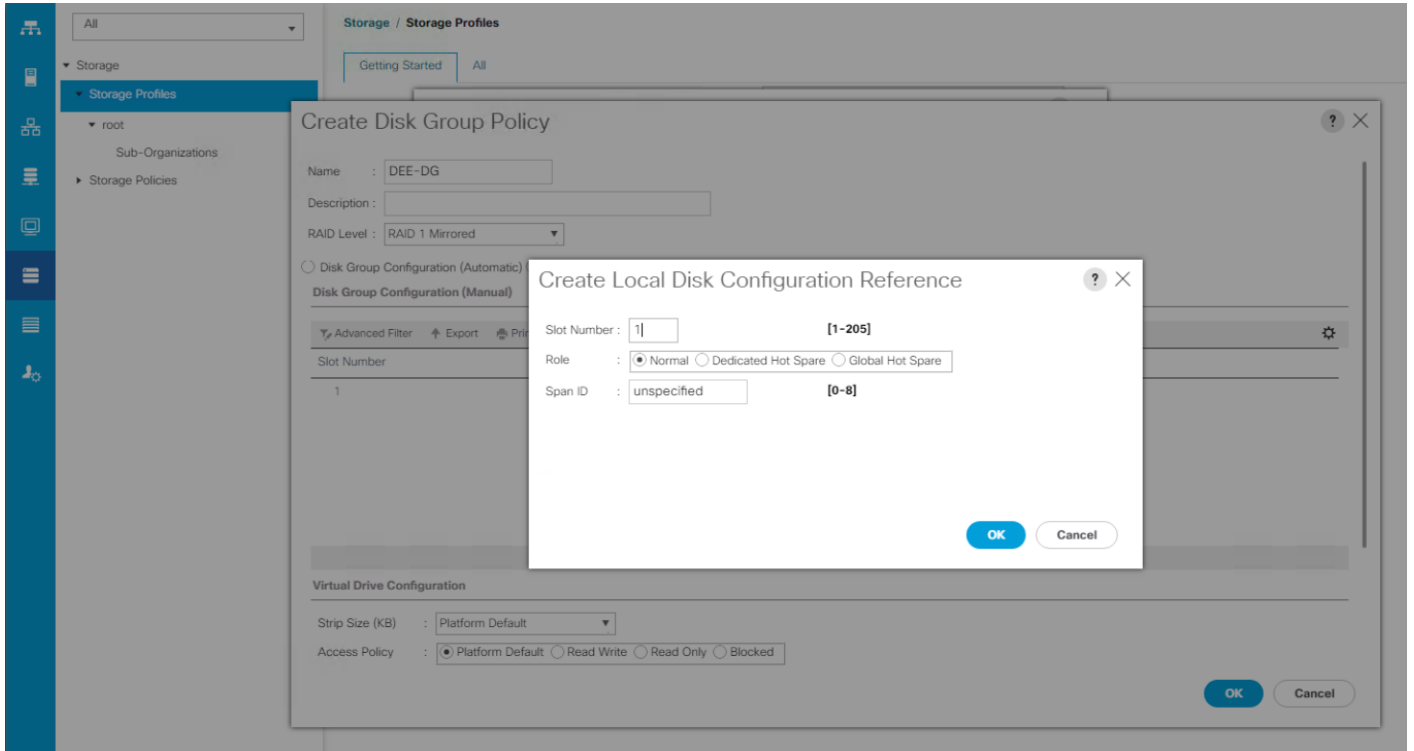
1. In Cisco UCS Manager, click Storage tab in the navigation pane.
2. Select Storage > Storage Profiles.
3. Right-click Storage Profiles and select Create Storage Profile.
4. Enter the name for the Storage Profile (for example, Docker-StgProf).



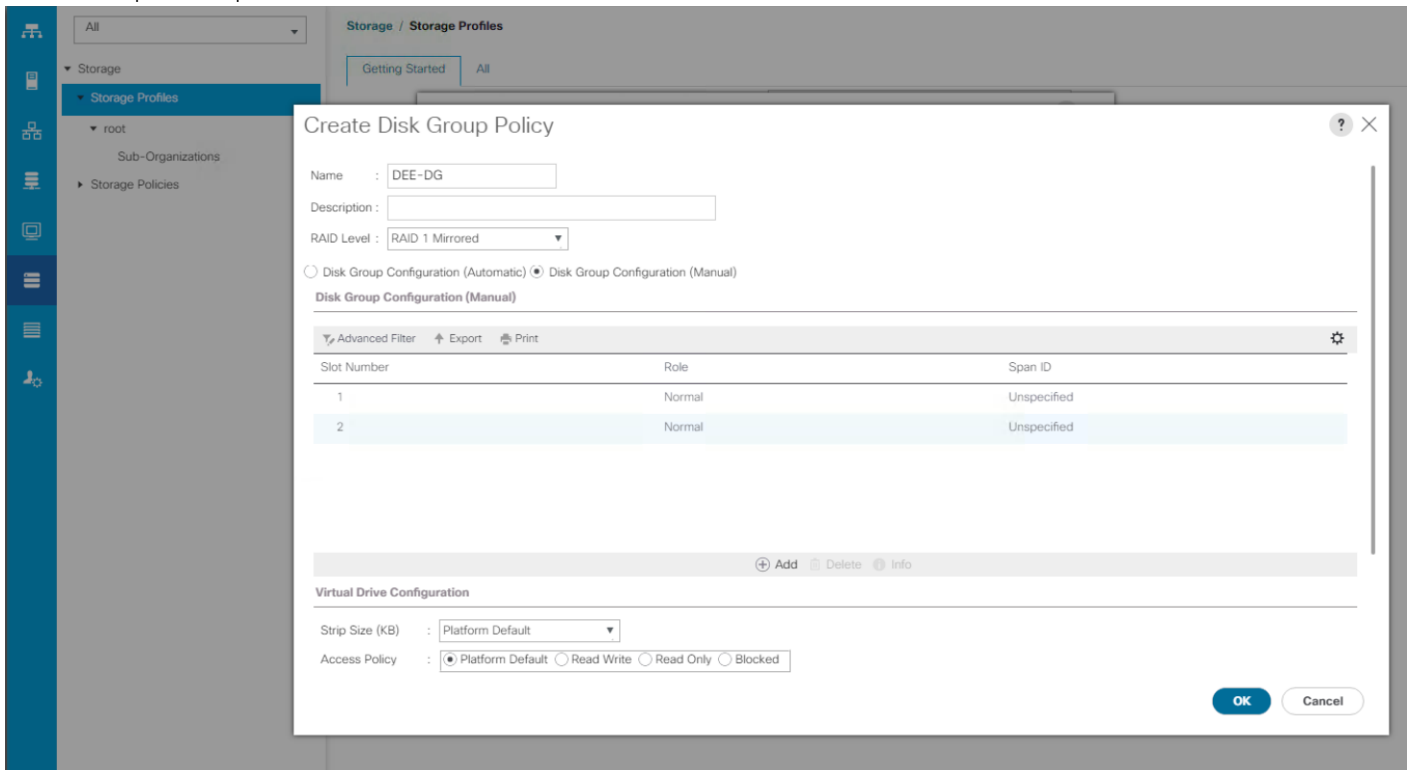
5. In the Local LUNs tab, click + on the right plane of the Create Storage Profile Window.
6. Create Local LUN window appears. Keep the Create Local LUN radio button selected.
7. Enter the LUN name (for example, Boot-LUN) and specify the desired size (for example, 60GB).



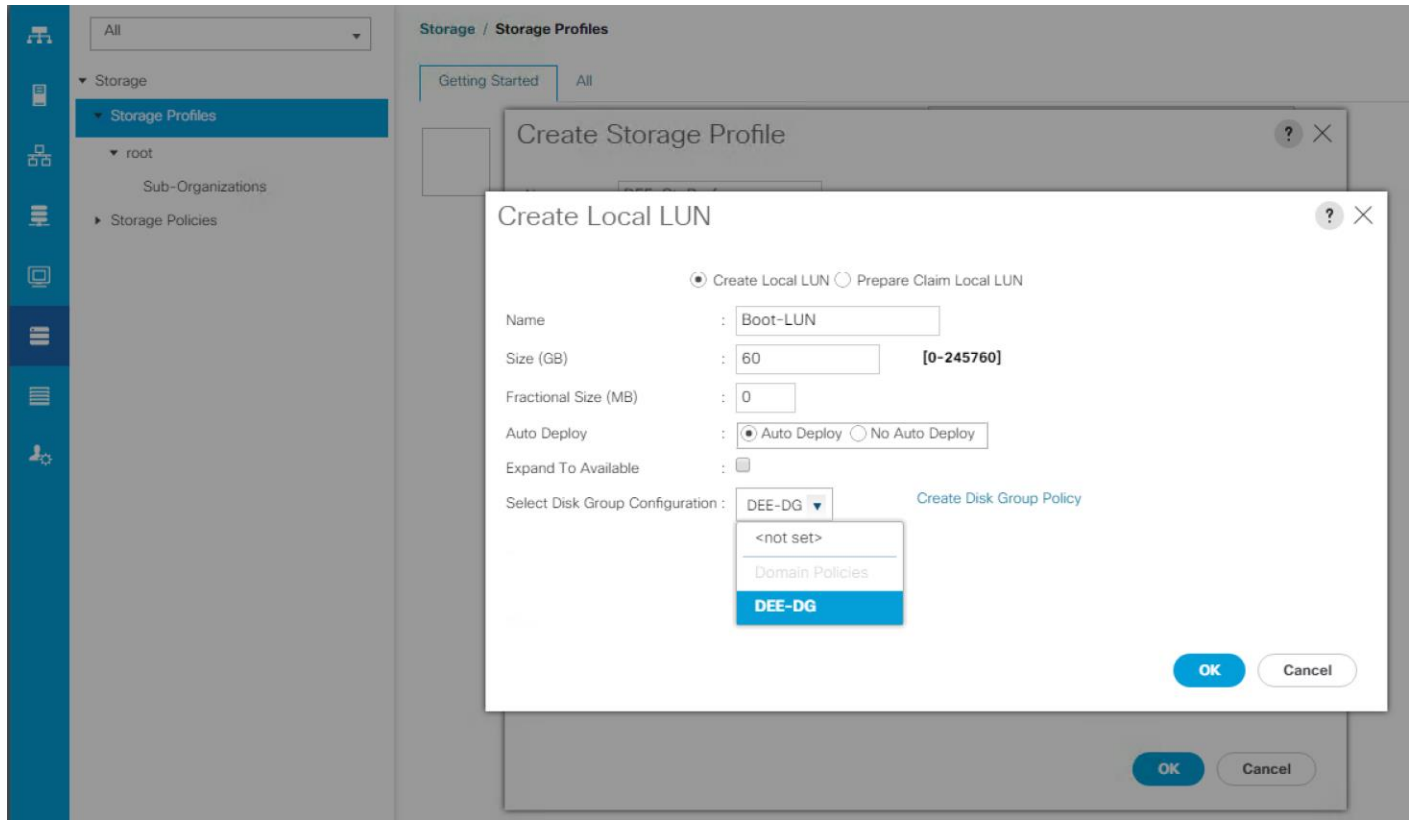
8. For select Disk Group configuration, Click Create Disk Group Policy.
9. Enter the Disk Group name (for example, Docker-DG). Keep the RAID Level as RAID 1 Mirrored, since blades come with only 2 disks.
10. Select Disk Group Configuration (Manual) radio button. Click + to add two slots; one for Boot-LUN and the other for Data-LUN. Keep the other fields as is and click OK.



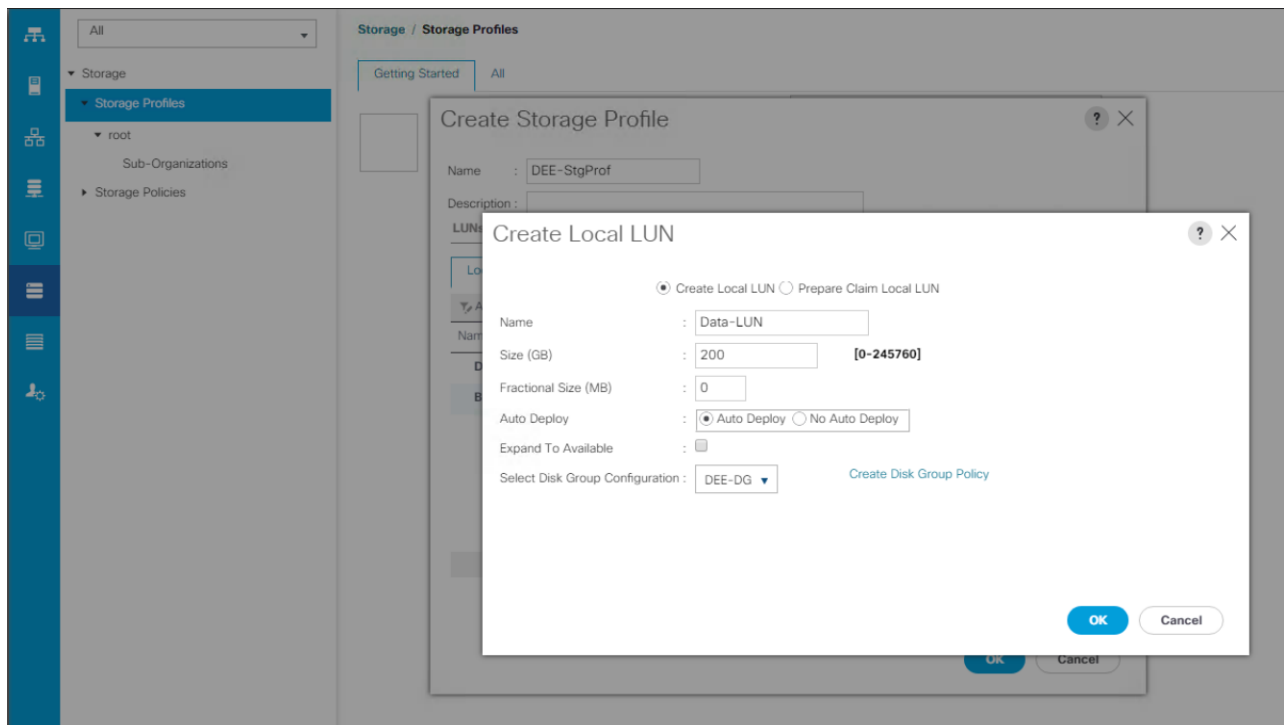
11. Repeat step 10 to select another slot which is 2 for blade servers. Click OK.



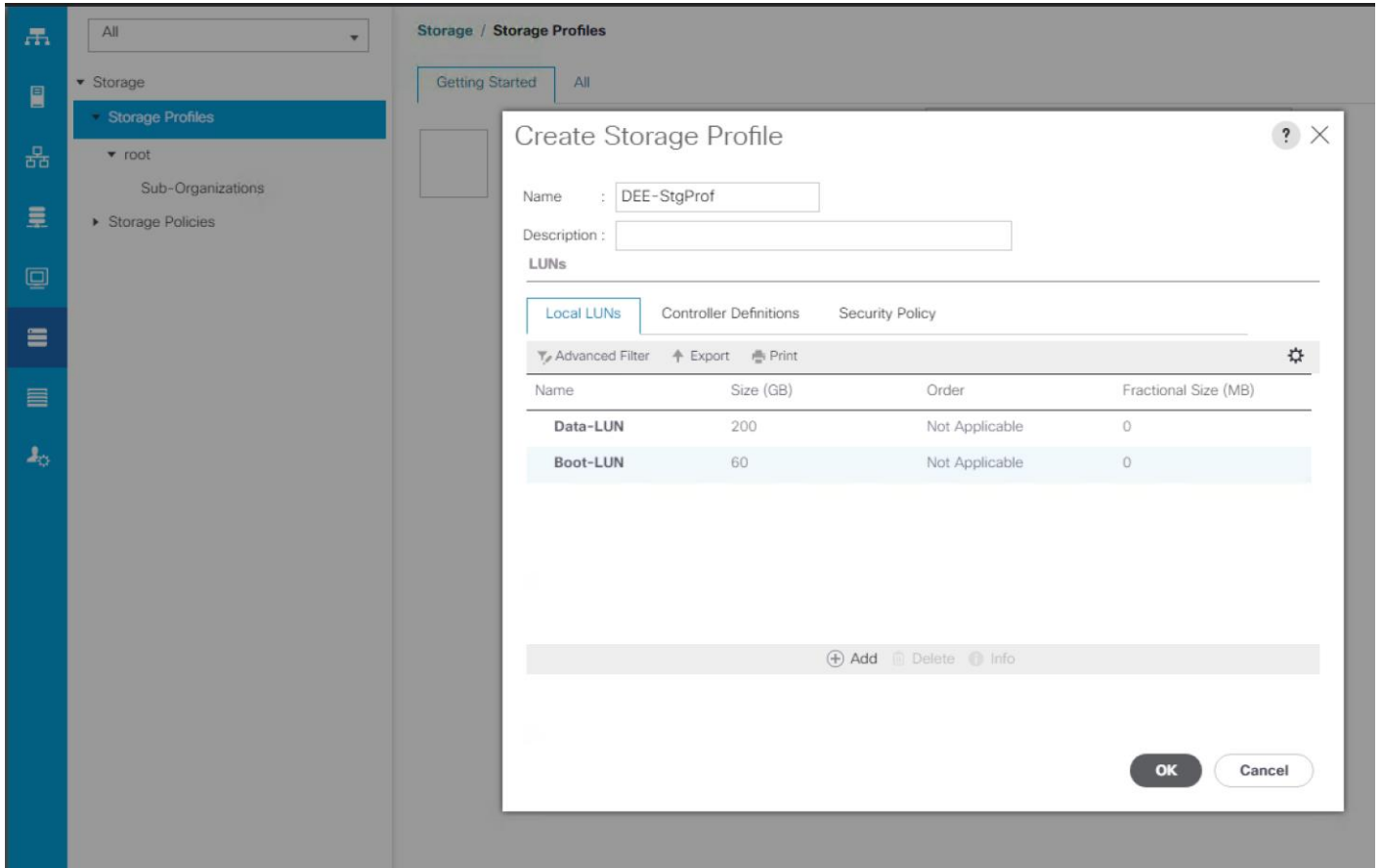
12. Select the created disk group from the Select Disk Group Configuration drop-down (for example, Docker-DG). Click OK to create Boot-LUN.



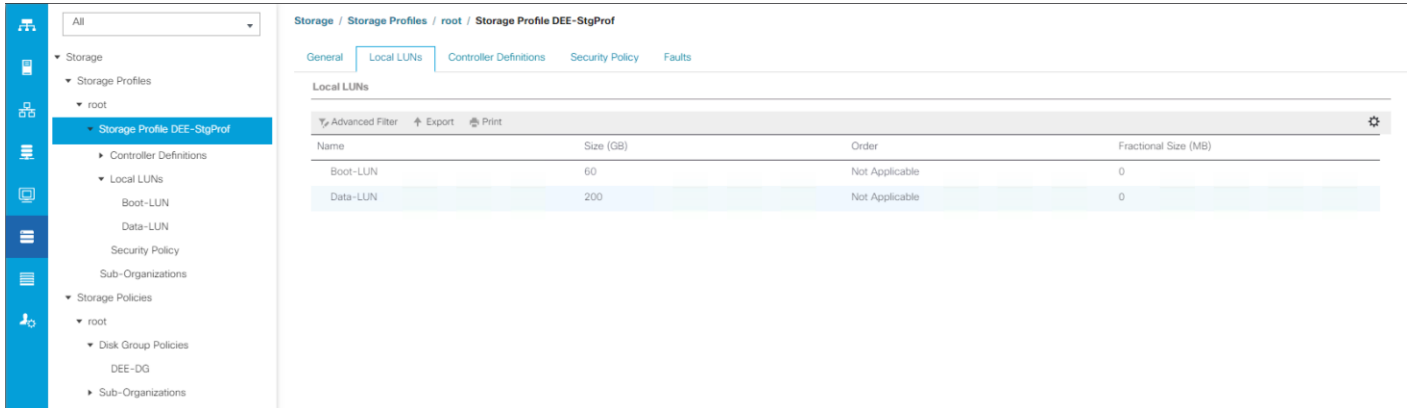
13. Repeat step 7 to create Data-LUN. Enter the appropriate name (for example, Data-LUN) and size (for example, 200GB) and select Disk Group Configuration (for example, Docker-DG). Click OK to create Data-LUN.



14. Press OK to create storage profile with 2 LUNs – Boot and Data.



15. The created Local LUNs can be view under Storage Profiles and Disk Group Policy under Storage Policies.





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

The screenshot shows the 'Storage / Storage Profiles / root / Storage Profile Docker' configuration page. The 'Local LUNs' tab is active, displaying a table with the following data:

Name	Size (GB)	Order	Fractional Size (MB)
Boot-Lun	100	Not Applicable	0
Data-Lun	3000	Not Applicable	0



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

The screenshot shows the 'Storage Policies / root / Disk Group Policies / Docker' configuration page. The 'General' tab is active, displaying the following configuration options:

- Name:** Docker
- Description:** [Empty field]
- RAID Level:** RAID 10 Mirrored And Stripe
- Disk Group Configuration:** Automatic (Manual is unselected)
- Disk Group Configuration (Automatic):**
 - Number of drives:** 6 [0-60]
 - Drive Type:** Unspecified (HDD, SSD are unselected)
 - Number of Dedicated Hot Spares:** unspecified [0-60]
 - Number of Global Hot Spares:** unspecified [0-60]
 - Min Drive Size (GB):** unspecified [0-10240]
 - Use Remaining Disks:**
 - Use JBOD Disks:** Yes (No is selected)
- Virtual Drive Configuration:**
 - Strip Size (KB):** Platform Default
 - Access Policy:** Platform Default (Read Write, Read Only, Blocked are unselected)
 - Read Policy:** Platform Default (Read Ahead, Normal are unselected)
 - Write Cache Policy:** Platform Default (Write Through, Write Back Good Sbu, Always Write Back are unselected)
 - IO Policy:** Platform Default (Direct, Cached are unselected)
 - Drive Cache:** Platform Default (No Change, Enable, Disable are unselected)
 - Security:**



RAID-10 provides mirrored and stripped pair of disks thereby giving redundancy and performance. RAID-10 requires a minimum of 4 disks, in our solution there are 6 disks on each of the C-Series servers. This gives us about 3TB of storage space for Docker run-time and local image store and allows greater storage scalability.

Creating Service Profile Templates

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

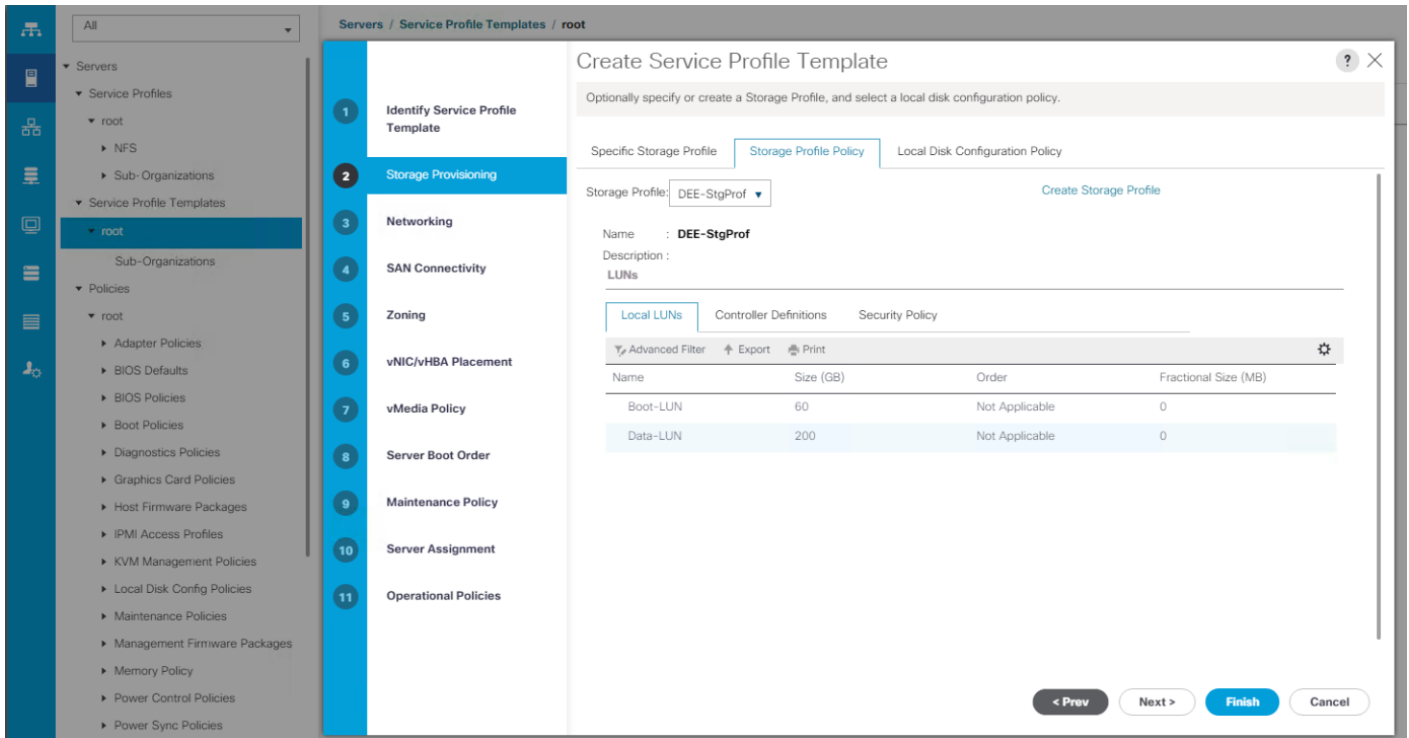
Creating Service Profile Template for UCP Manager/Master Nodes

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

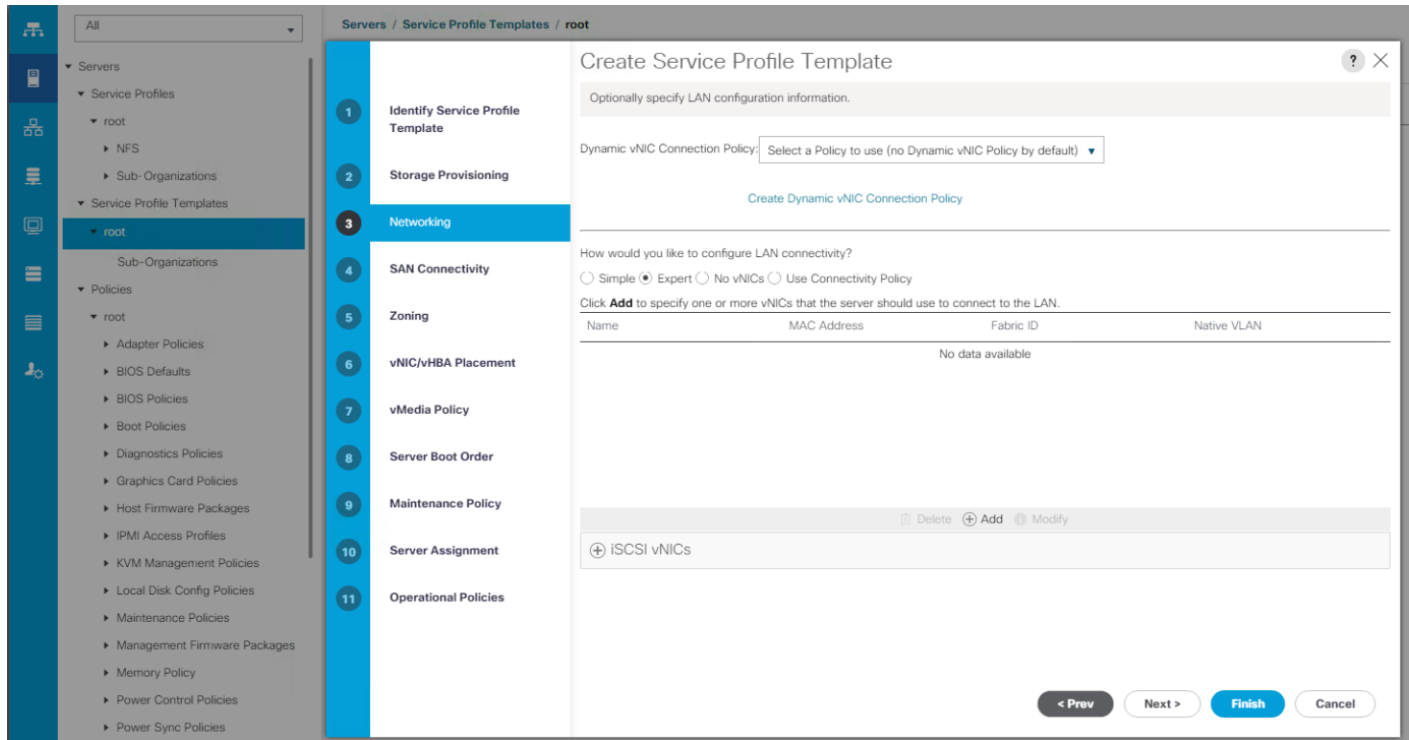
1. From Cisco UCS Manager, click Servers tab in the navigation pane.
2. Select Servers > Service Profile Template > root.
3. Right-click root and select Create Service Profile Template to open the Create Service Profile Template wizard.
4. In the Identify the Service Profile Template screen, configure the following:
 - a. Enter name (for example, DEE-Ctrl) 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.

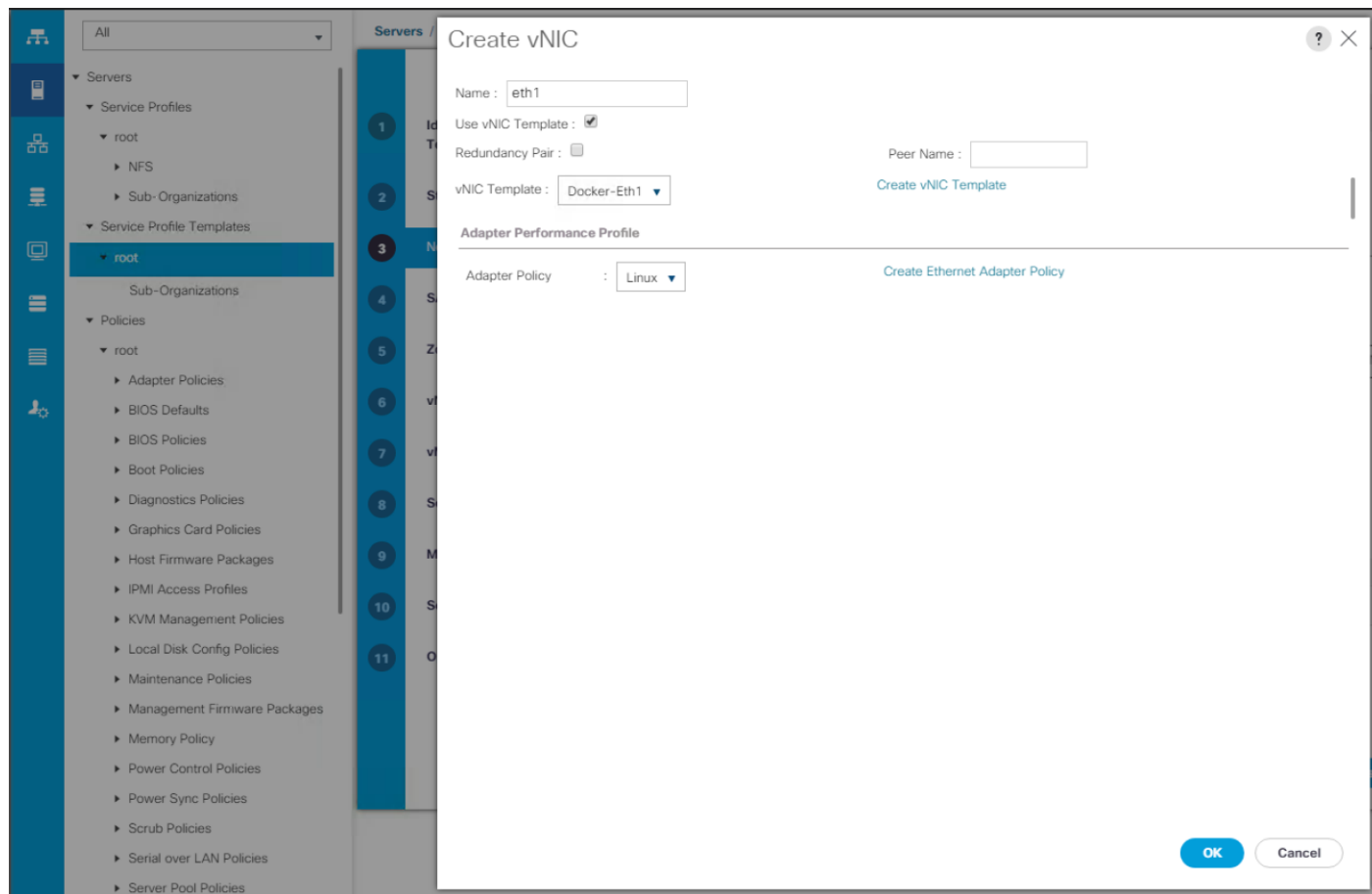
- b. In the Storage profile drop-down, select a policy. Choose the previously configured policy (for example, Docker-StgProf). Local LUNs tab lists the previously configured Local LUNs.
- c. Click Next.



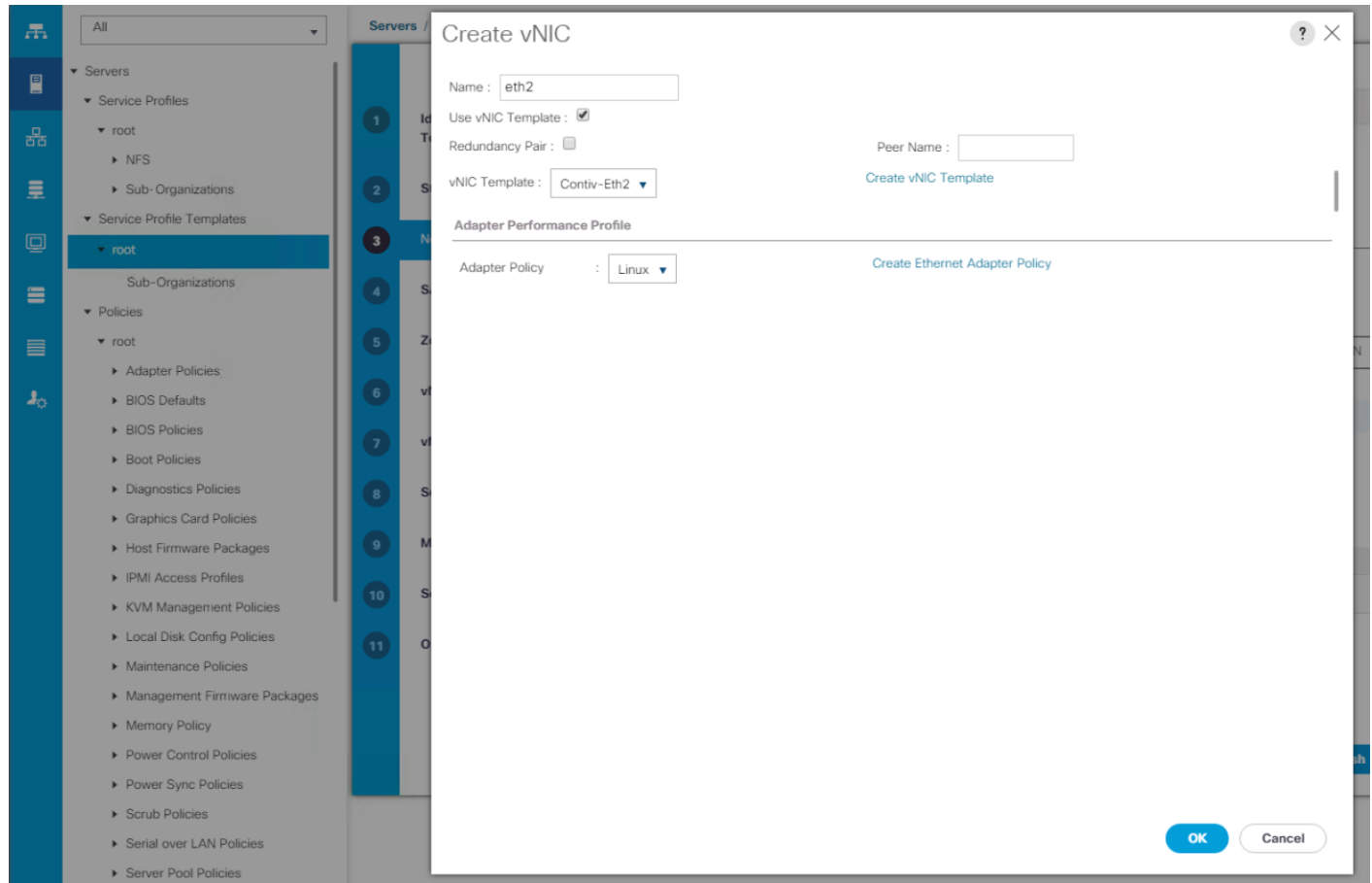
6. In the Networking screen, configure the following:
 - a. Restore the default setting for Dynamic vNIC Connection Policy.
 - b. Click Expert radio button to configure the LAN connectivity.



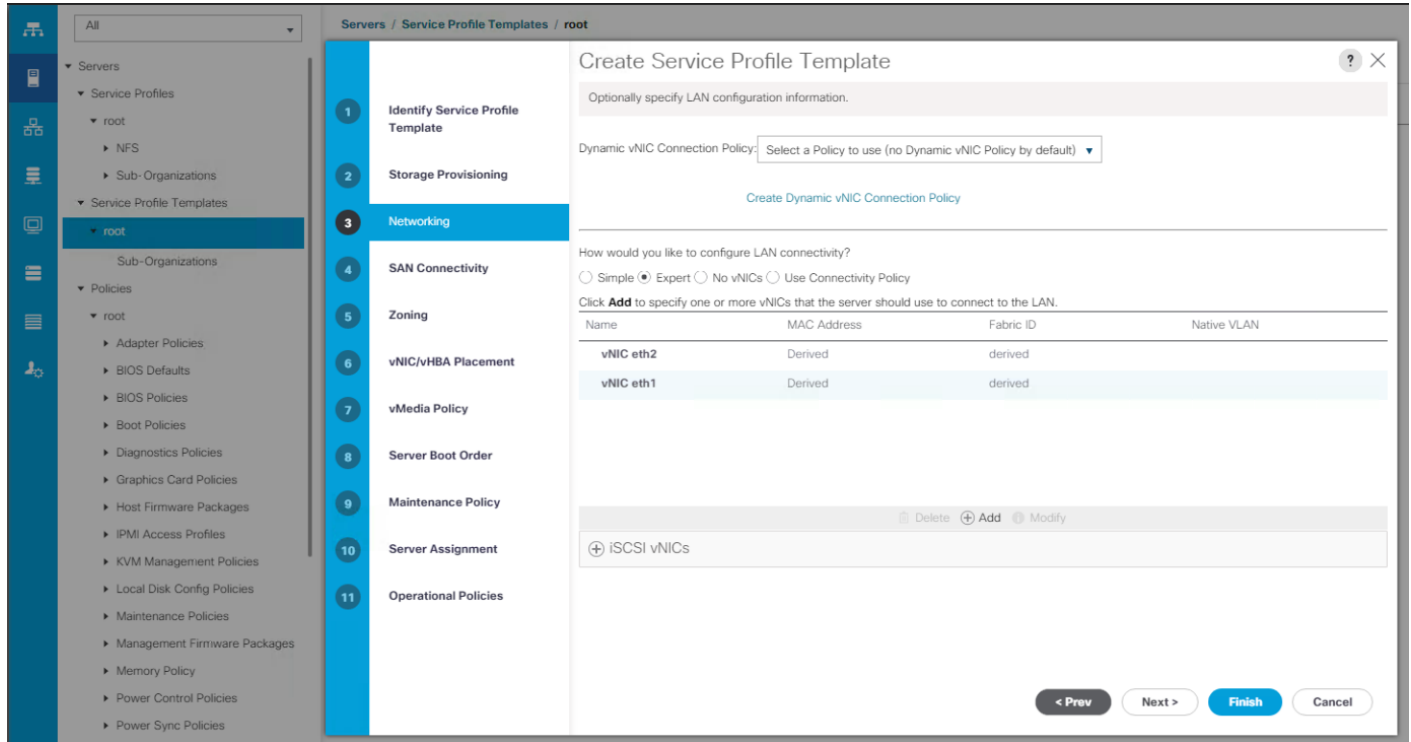
- c. Click on [+ Add], to add a vNIC to the template.
- d. In the Create vNIC dialog box:
 - Enter the name (for example, eth1) of the vNIC.
 - Check the Use vNIC Template check box.
 - In the vNIC Template list, choose the previously created vNIC Template for Fabric A boot (for example, Docker-Eth1).
 - In the Adapter Policy list, choose Linux.
 - Click OK to add this vNIC to the template.



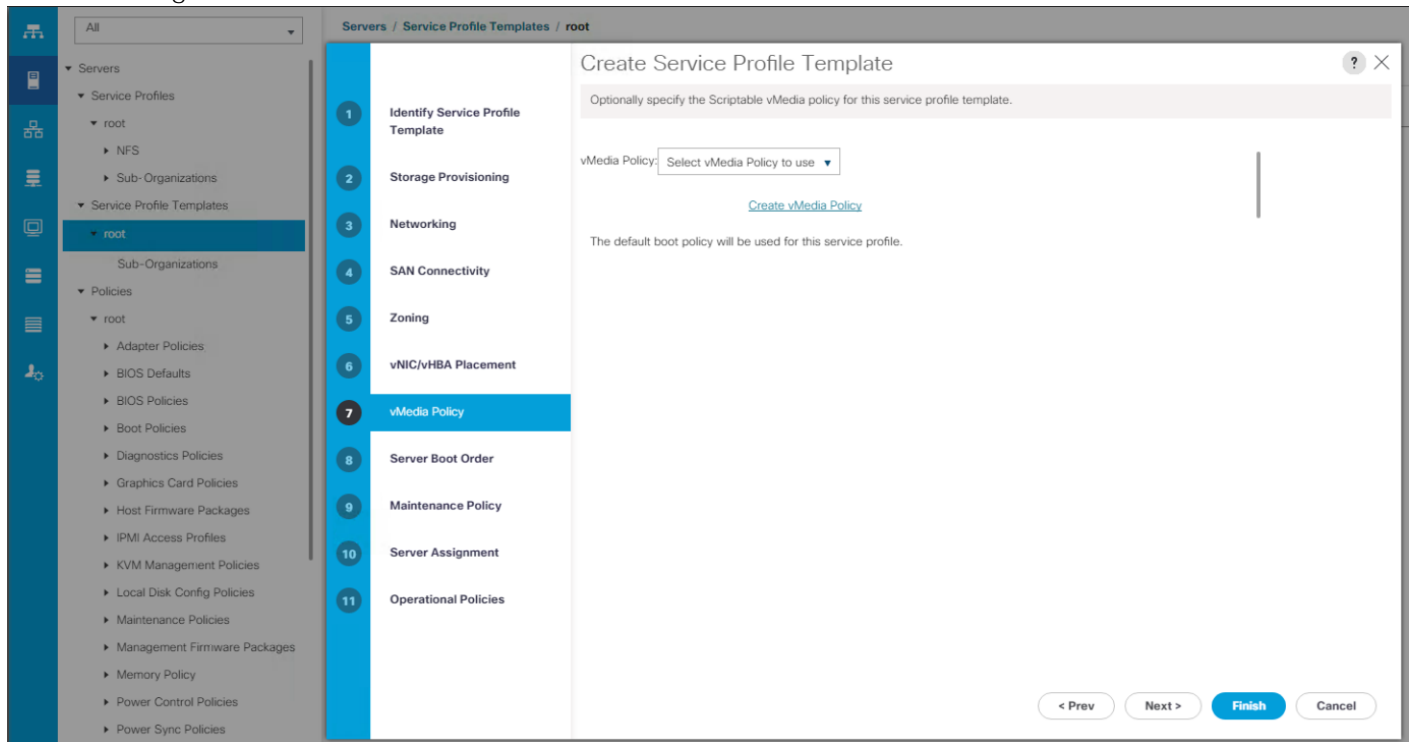
- e. Click on [+] Add to add a second vNIC to the template.
- f. In the Create vNIC dialog box:
 - Enter the name (for example, eth2) of the vNIC.
 - Check the Use vNIC Template check box.
 - In the vNIC Template list, choose the previously created vNIC Template for Fabric B boot (for example, Contiv-Eth2).
 - In the Adapter Policy list, choose Linux.
 - Click OK to add this vNIC to the template.



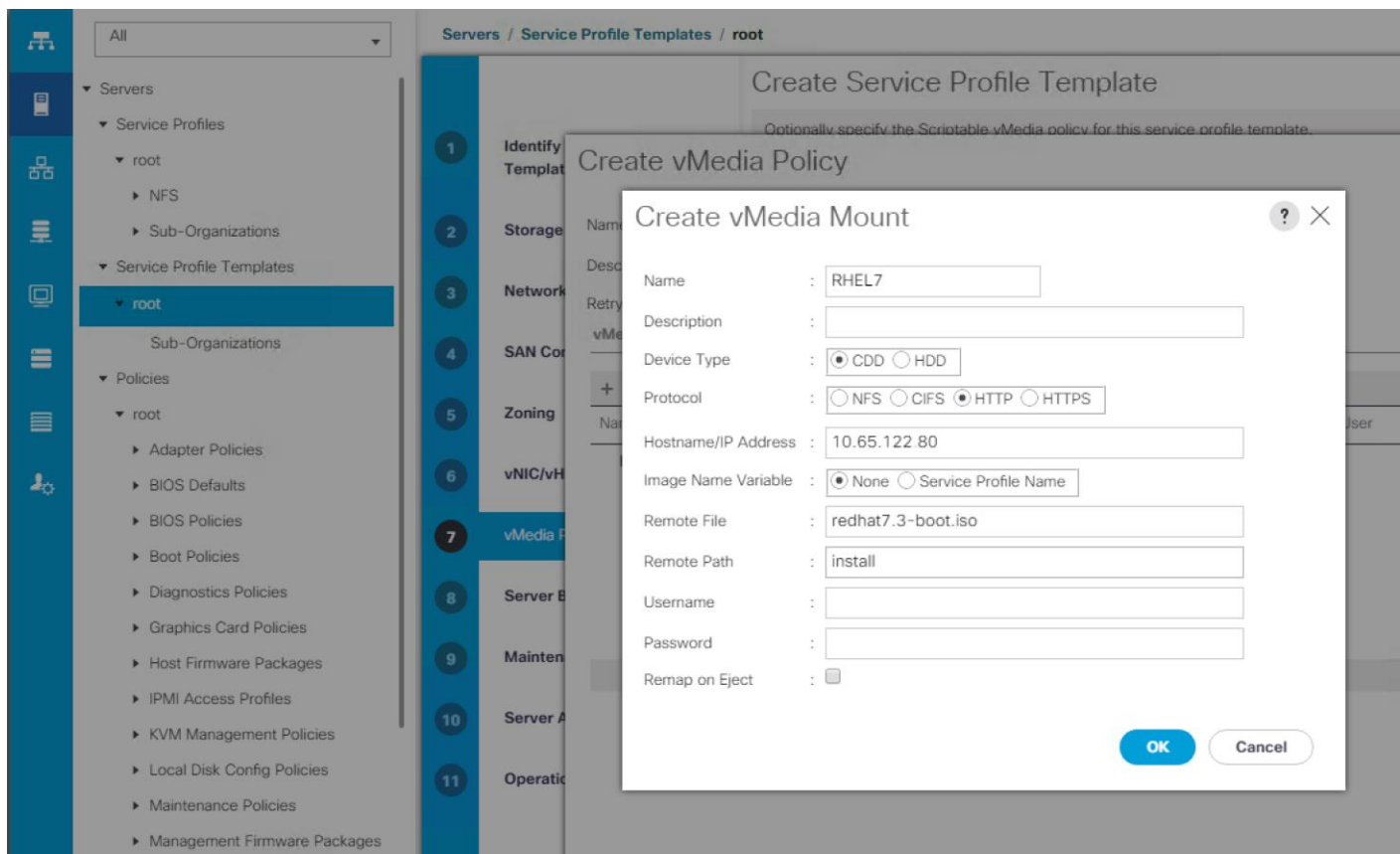
- g. Review the configuration on the Networking screen of the wizard. Make sure that both the vNICs are created. Click Next.



7. Click Next in the SAN Connectivity after selecting `No vHBAs`, skip Zoning and vNIC/vHBA Placement sections by pressing Next.
8. Under vMedia Policy, click Create vMedia Policy. This policy will enable us to install bare metal operating system using PXE less automated environment through scripted vMedia policy feature in UCS Manager.



9. Name vMedia policy, keep `Retry on Mount Failure` to default, which `yes`. Click on [+] Add for adding CIMC mounted media –
 - a. Mount point name can be of your choice, RHEL7 is used in this guide, as the solution is based on RHEL7.x OS version.
 - b. Device type for boot images should be set to `CDD` as an ISO image will be mounted during install time.
 - c. Protocol should be set to HTTP, as kernel and installer configuration are hosted on web-server kick-start file.
 - d. Hostname/IP Address of the HTTP server host.
 - e. For boot kernel images, set Image Name Variable as `None`.
 - f. ISO file name which will be mounted for installing boot images.



Create vMedia Policy

Name :

Description :

Retry on Mount Failure : No Yes

vMedia Mounts

Name	Type	Protocol	Authentica...	Server	Filename	Remote Path	User	Remap on ...
RHEL7	CDD	HTTP	Default	10.65.122...	redhat7.3-...	install		No

+ - Advanced Filter Export Print

+ Add Delete Info

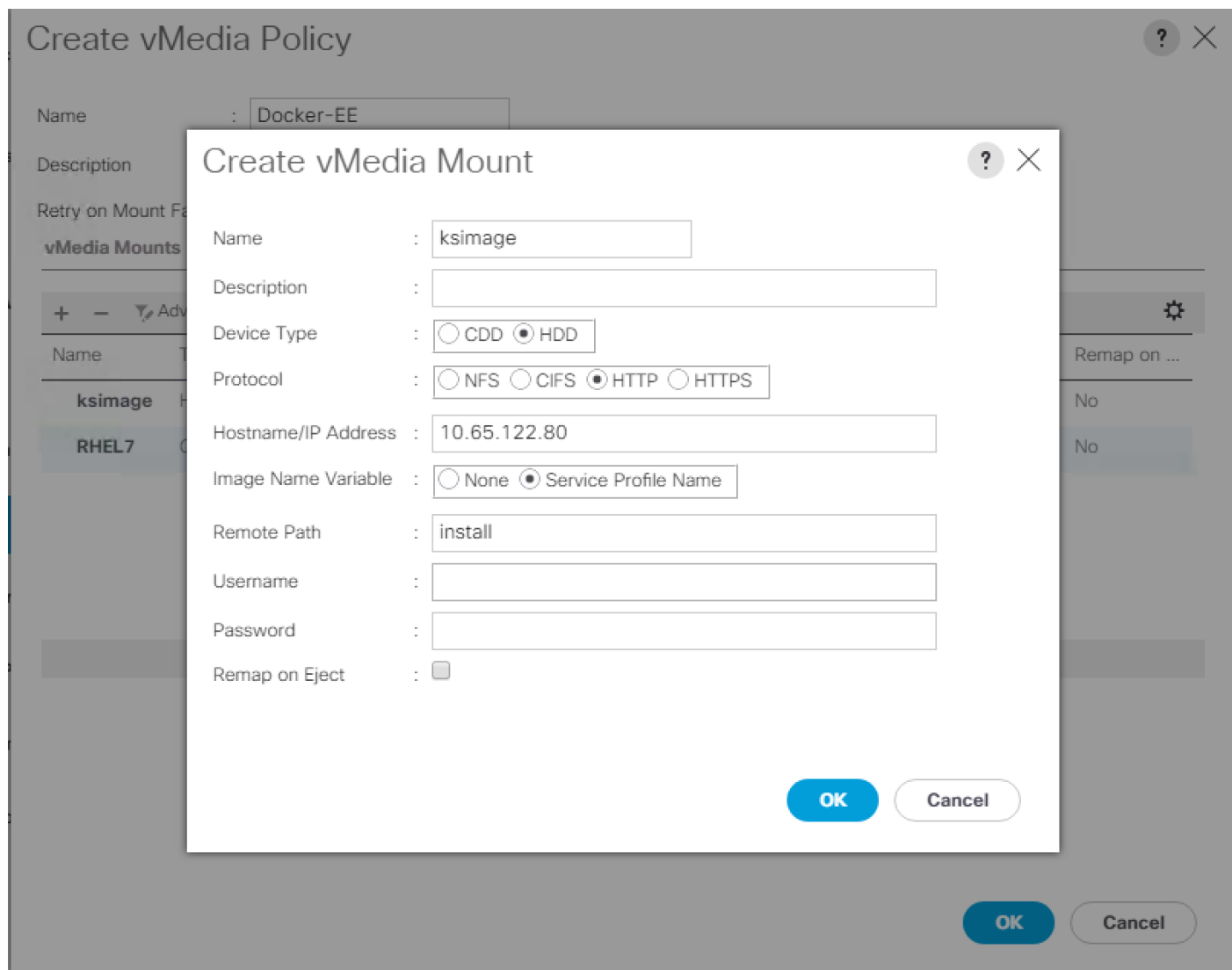
OK Cancel

10. Press OK to move forward to configure mounting of Kickstart file. Press [+] Add again.
 - a. Use `ksimage` as the mount point name.
 - b. Select device type for second mount as `HDD`.
 - c. Keep the protocol as `HTTP`, as this image is hosted on the same server as for the first mount point.
 - d. Host address is same as before `10.65.122.80`, which is the web-server hostname.
 - e. Image variable name should be selected as `Service Profile Name`. This will enable you to customize kickstart file for each Docker EE cluster nodes. Setting ip address and hostname for individual nodes are the two major items for customization.



Here kickstart file must be named to match with the service profile names of the cluster nodes.

- f. Remote path should be the same as before - `install`. This is the directory served by the web-server containing boot kernel images and kickstart files.



11. Click OK to continue finishing vMedia policy creation.

Create vMedia Policy



Name : Docker-EE

Description :

Retry on Mount Failure : No Yes

vMedia Mounts

Name	Type	Protocol	Authentic...	Server	Filename	Remot...▲	User	Remap o...
ksim...	HDD	HTTP	Default	10.65.122.80		install		No
RHEL7	CDD	HTTP	Default	10.65.122.80	redhat7.3-boot.iso	install		No

Add
 Delete
 Info

OK

Cancel

12. Select the vMedia policy that was created in the service profile template as shown below. Click Next.

Create Service Profile Template

Optionally specify the Scriptable vMedia policy for this service profile template.

vMedia Policy: Docker-EE

[Create vMedia Policy](#)

Name : **Docker-EE**
 Description :
 Retry on Mount Failure : **Yes**

vMedia Mounts

Name	Type	Protocol	Authenticat...	Server	Filename	Remote Path	User	Remap on ...
ksimage	HDD	HTTP	None	10.65.122....		install		No
RHEL7	CDD	HTTP	None	10.65.122....	redhat7.3-...	install		No

< Prev Next > **Finish** Cancel

13. In the Set Boot Order screen, select the previously created boot policy from the Boot Policy drop-down (for example, DEE-vMedia).

Create Service Profile Template

Optionally specify the boot policy for this service profile template.

Select a boot policy.

Boot Policy: **DEE-vMedia** [Create Boot Policy](#)

Name : **DEE-vMedia**
 Description :
 Reboot on Boot Order Change : **No**
 Enforce vNIC/vHBA/iSCSI Name : **Yes**
 Boot Mode : **Legacy**

WARNINGS:
 The type (primary/secondary) does not indicate a boot order presence.
 The effective order of boot devices within the same device class (LAN/Storage/iSCSI) is determined by PCIe bus scan order.
 If **Enforce vNIC/vHBA/iSCSI Name** is selected and the vNIC/vHBA/iSCSI does not exist, a config error will be reported.
 If it is not selected, the vNICs/vHBAs are selected if they exist, otherwise the vNIC/vHBA with the lowest PCIe bus scan order is used.

Boot Order

Name	Order	vNIC/vHB...	Type	WWN	LUN Name	Slot Numb...	Boot Name	Boot Path	Description
Local L...	1								
CIMC ...	2								
CIMC ...	3								

[Create iSCSI vHBC](#) [Set iSCSI Boot Parameters](#) [Set USB Boot Parameters](#)

< Prev Next > **Finish** Cancel

14. Click Next.

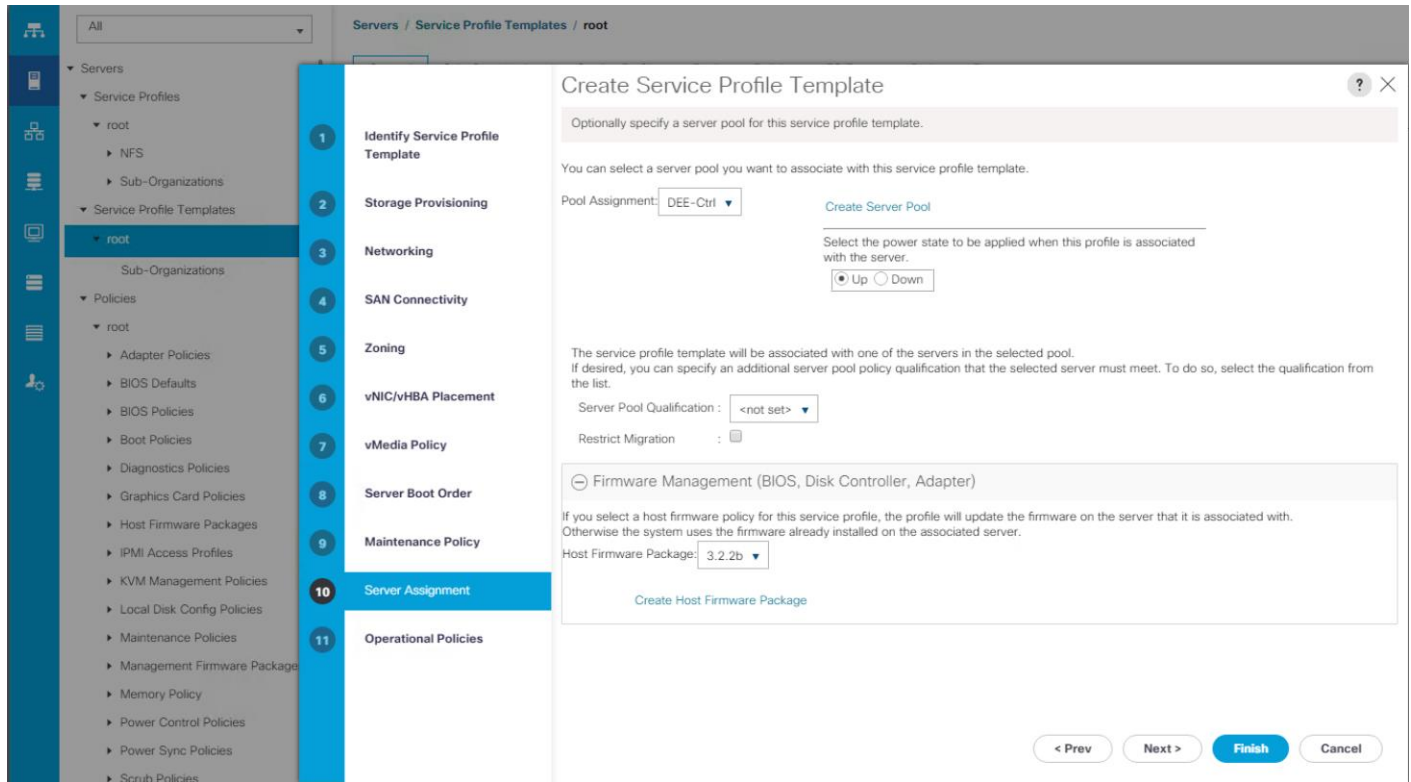
15. Click Next in Maintenance Policy screen after selecting maintenance policy as `default` .

16. In the Server Assignment screen, configure the following:

- For Pool Assignment, choose the previously created server pools from the list (for example, DEE-Ctrl).
- Leave the Power State as UP for when the Profile is applied to a server
- For Server Pool Qualification, select the previously created policy from the list (for example, all-chassis).
- Expand the Firmware Management section. For the Host Firmware Package, select the previously selected policy from the list (for example, 3.2.2b).
- Click Next.

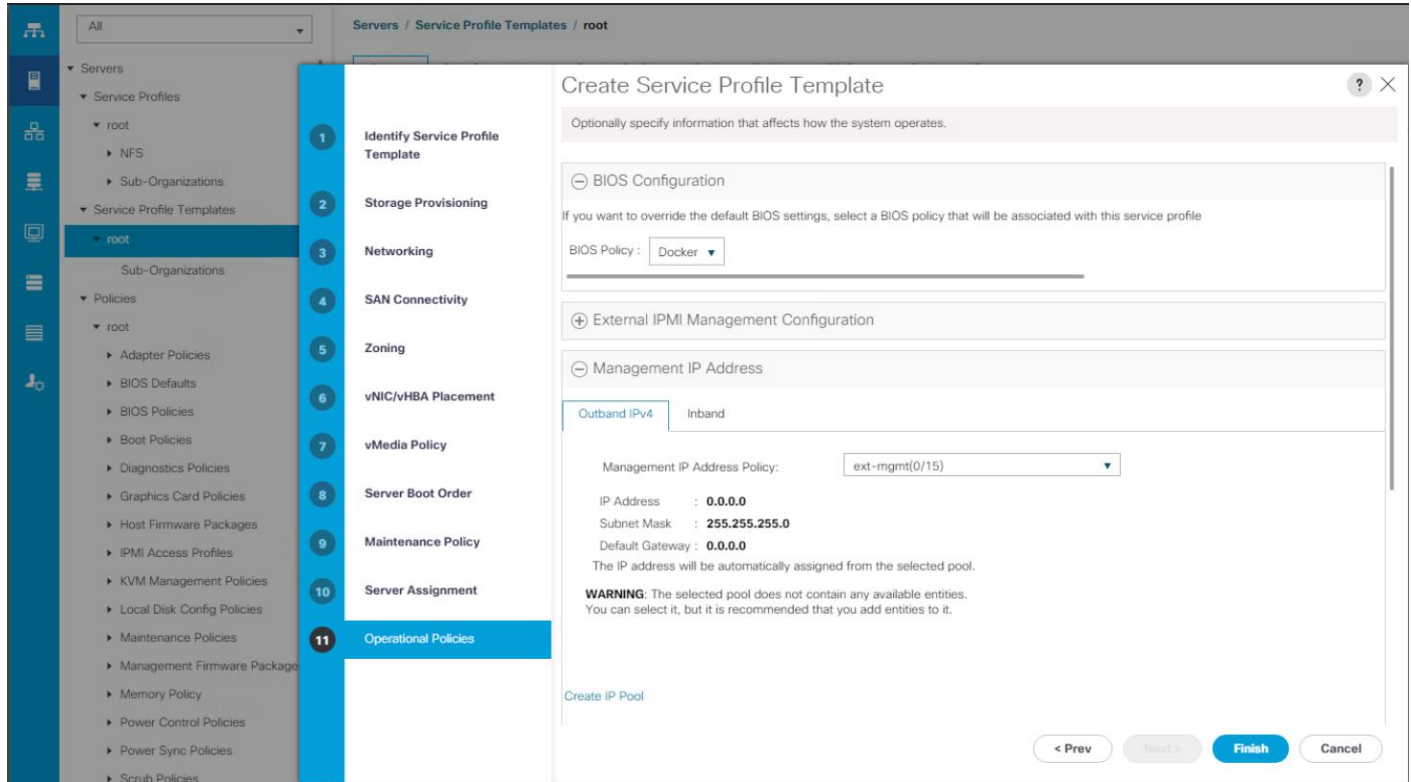


Assigning template to server pool results in automatic association of service profiles whenever they get instantiated from the templates.

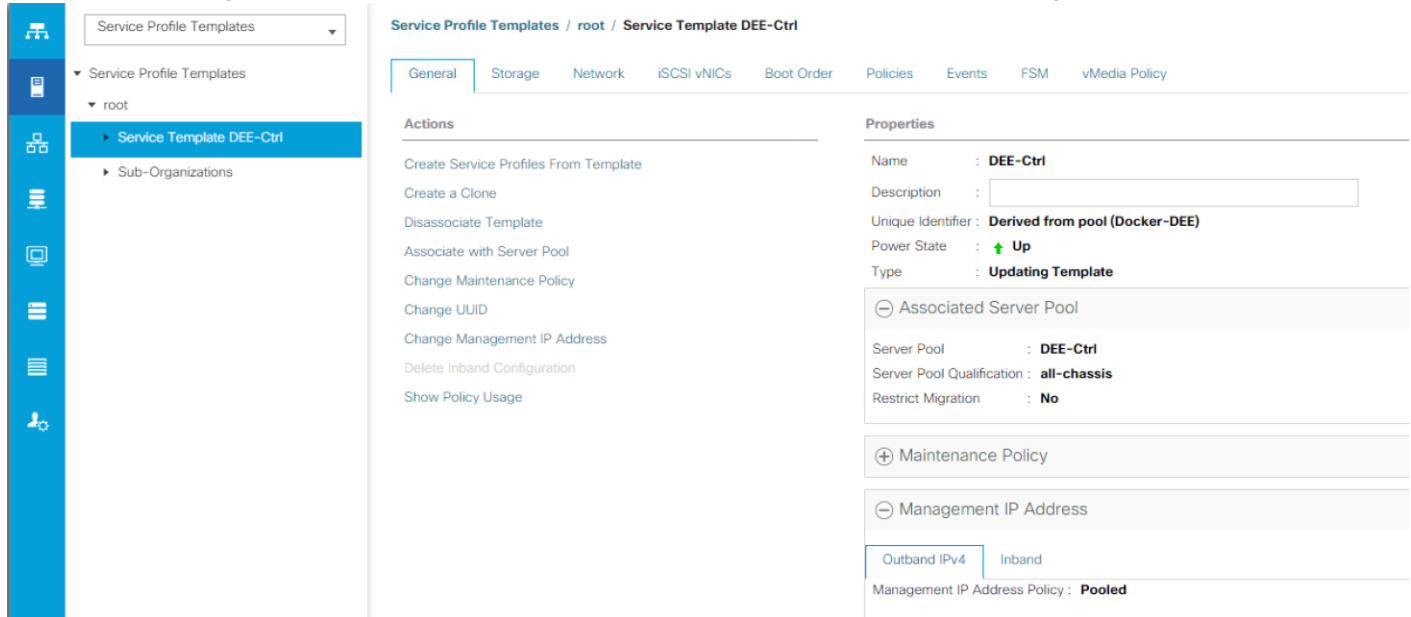


17. In the Operation Policies screen, configure the following:

- a. For the BIOS Policy list, select the previously configured policy (for example, Docker).
- b. Expand Management IP Address, select the IP address pool (for example, ext-mgmt (0/15)) from the management IP Address policy down-down under outband IP4 option.



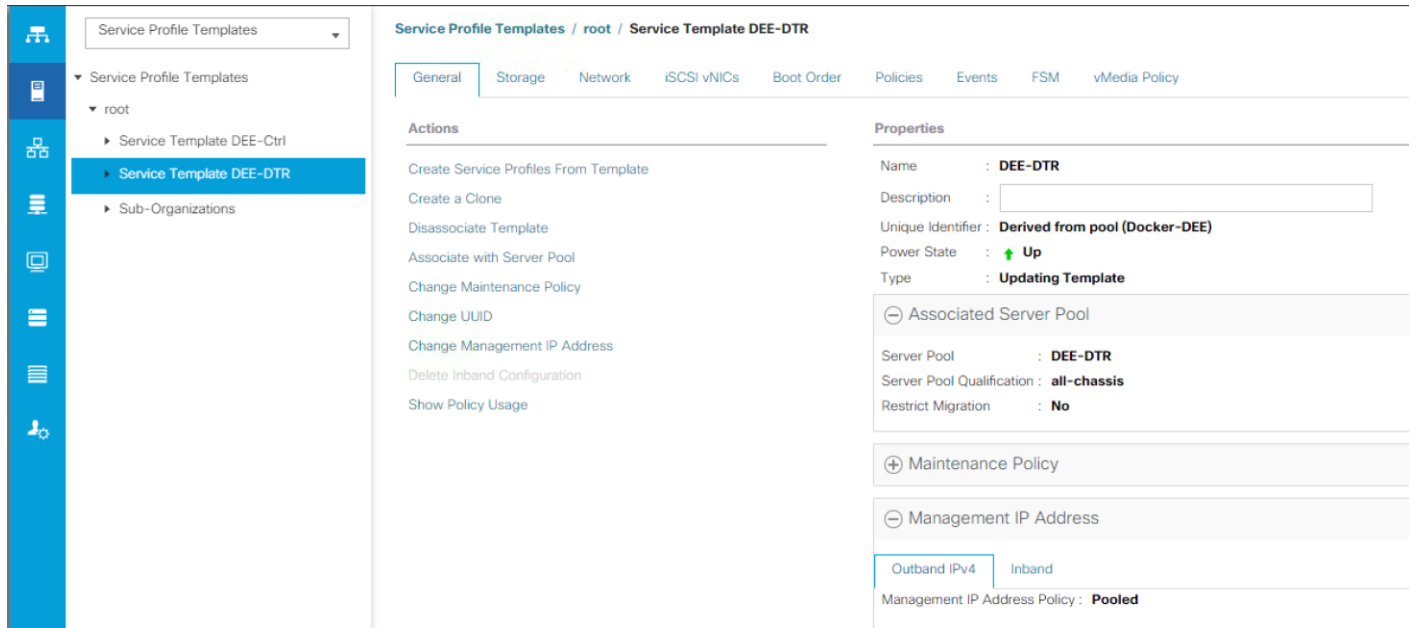
18. Click Finish to complete the creation of the Service Profile Template. Created service profile template will get listed under Service Profile Templates as shown in the below figure.



Creating Service Profile Template for DTR Nodes

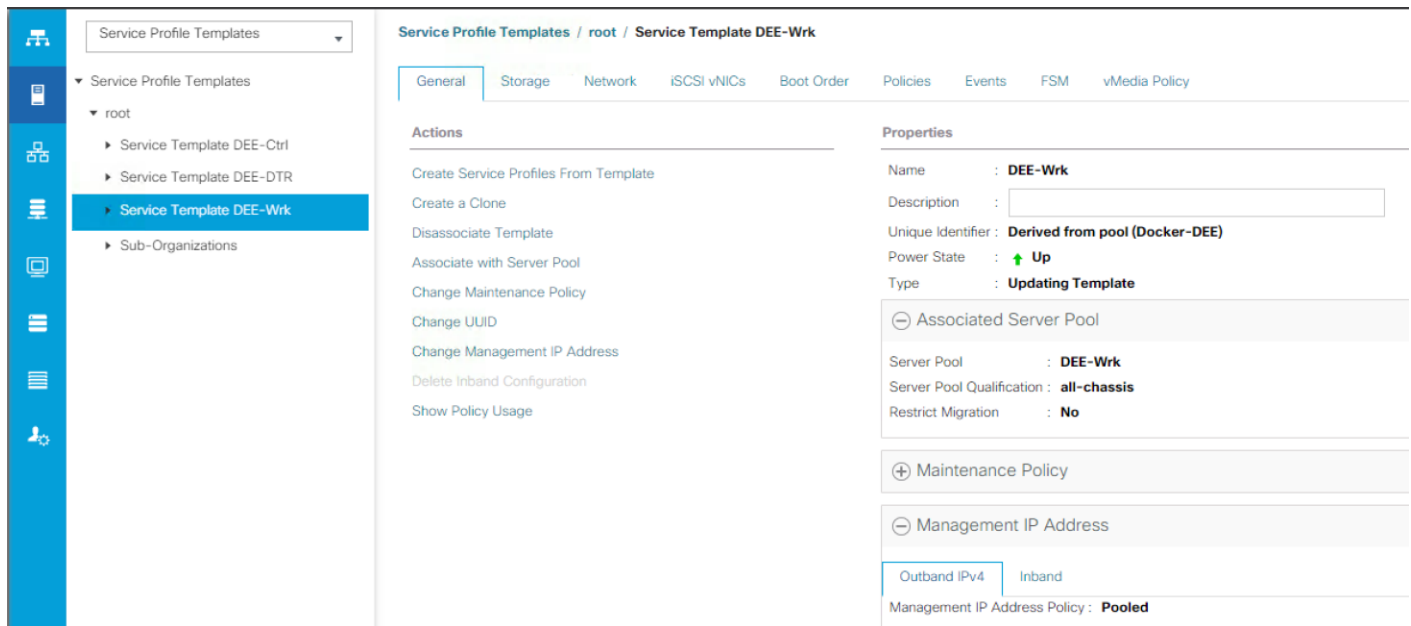
Repeat the steps 1 to 18 detailed in the previous sub-section for creating a service profile template for DTR nodes using already created policies and pools previously. For example, vMedia policy Docker-EE and boot

policy DEE-vMedia. After creating the service profile template, the template for DTR nodes will look similar to the below screenshot.



Creating Service Profile Template for UCP Worker Nodes

Repeat the steps 1 to 18 detailed in the previous sub-section for creating a service profile template for UCP worker nodes using already created policies and pools previously. For example, vMedia policy Docker-EE and boot policy DEE-vMedia. After creating the service profile template, the template for UCP worker nodes will look similar to the below screenshot.



Configuring PXE-less Automated OS Installation Infra with UCSM vMedia Policy

UCSM vMedia policy applied with CIMC mounted vMedia devices in boot policy give us a way to creates an automated operating system installation infrastructure for DEE cluster bare metal nodes. This does not require combination of PXE and TFTP servers for doing the same job. This section provides details on configuring vMedia policy to create an automated environment to provision bare metal Red Hat 7.3 installations required for Docker EE cluster nodes.



PXE-less automated OS installation infra with UCS Manager vMedia policy is not an essential deployment step for this solution. It is being documented here with an intention to provide users a quick and effective way to automate bare metal provisioning especially at scale. Users of this guide are free to choose any of the methods they are familiar with, including manual method using CD/DVD.

Prerequisites

To use UCS Manager vMedia policy for automated provisioning of operating system following essential prerequisites are needed:

1. A build server with Red Hat OS version 7 and above is needed. This solution uses a RHEL 7.3 host.



Build server is not part of the solution BOM, as users/administrators are free to use any of the existing RH based system which meets the requirements.

2. Web service, which can be run on the same build server, with bare minimum configuration to host and serve boot kernel ISO, kickstart configuration image files and installation media.
3. Web services should be running on the same management vLAN as that of the UCS Manager. It should also be configured with IP address from the same subnet as that of UCS Manager VIP.



If having a web-server on the same vLAN and subnet as that of UCS Manager can be a challenge, then it should at least run a subnet having inter-vLAN routing configured for UCS Manager's subnet. Web-server should be accessible from the UCS Manager subnet. However, for an optimal performance on automated installation, it is recommended to have a web-server and UCS Manager on the same subnet.

Web Server – Installation and Configuration

1. On a server designated as build server install Apache http server. It can either be installed at the time of build server provisioning or post to it by issuing following command:

```
# yum install http
<snip>
Running transaction
  Installing : mailcap-2.1.41-2.e17.noarch 1/3
  Installing : httpd-tools-2.4.6-45.e17_3.4.x86_64 2/3
  Installing : httpd-2.4.6-45.e17_3.4.x86_64 3/3
  Verifying  : httpd-tools-2.4.6-45.e17_3.4.x86_64 1/3
  Verifying  : mailcap-2.1.41-2.e17.noarch 2/3
  Verifying  : httpd-2.4.6-45.e17_3.4.x86_64 3/3

Installed:
httpd.x86_64 0:2.4.6-45.e17_3.4
Dependency Installed:
```

```
httpd-tools.x86_64 0:2.4.6-45.el7_3.4
mailcap.noarch 0:2.1.41-2.el7
```

Complete!

- Open /etc/httpd/conf/httpd.conf in an editor and change following parameters:

```
Listen <ip address of the server>:80
ServerName <ip address of the server>:80
```

- Enable and start httpd.service:

```
# systemctl enable httpd.service
# systemctl start httpd.service
# firewall-cmd --zone=public --permanent --add-service=http
```

- Create directories inside http document root

```
# mkdir /var/www/html/ISO
# mkdir /var/www/html/vMedia
# mkdir /var/www/html/install
```

- Comment out all lines inside /etc/httpd/conf.d/welcome.conf




```
#<LocationMatch "^/+$">
#   Options -Indexes
#   ErrorDocument 403 /.noindex.html
#</LocationMatch>
#
#<Directory /usr/share/httpd/noindex>
#   AllowOverride None
#   Require all granted
#</Directory>
#
#Alias /.noindex.html /usr/share/httpd/noindex/index.html
```

- Restart httpd.service

```
# systemctl restart httpd.service
```

- Web server should show following directory structure, that shown earlier

Index of /

	Name	Last modified	Size	Description
	ISO/	2017-12-02 04:10	-	
	install/	2017-12-02 04:10	-	
	vMedia/	2017-12-02 04:10	-	

Create Images

Two images are needed for bootstrapping the operating system on the bare metal DEE cluster nodes:

- ISOLINUX Boot ISO Image - This image is common for all the DEE cluster nodes. To prevent clogging the network at one go with entire OS distribution while bootstrapping the cluster nodes, bare minimal

files and configs along with vmlinuz and initrd images are part of this boot IOS image. Rest of the installation media is served separately.

2. Kickstart Disk Image – This consists of kickstart file only, to be mounted on the bootstrapped nodes via vMedia policy. This unique disk image is created for each cluster nodes and named as the UCSM service profile names of each of them. Each disk image differs with each other only by host-names and IP address of the bare metal nodes getting installed with operating system. Rest of the configuration remains same.

Creating Boot ISO Image

The boot ISO image requires specific binaries and other files to be extracted out of OS installation media.

1. Copy RHEL 7.3 installation DVD ISO file to the root of the build/web-server
2. Mount ISO to the web-**server's /ISO document root**

```
# mount -o loop RHEL-7.3-20161019.0-Server-x86_64-dvd.iso /var/www/html/ISO/
```
3. Extract following files from mounted ISO to /var/www/html/vMedia


```
# cd /var/www/html/vMedia/
# cp -a ../ISO/isolinux .
# cp ../ISO/.discinfo isolinux/
# cp ../ISO/.treeinfo isolinux/
# cp -a ../ISO/LiveOS isolinux/
# cp -a ../ISO/images isolinux/
# chmod 664 isolinux/isolinux.bin
```
4. Edit isolinux/isolinux.cfg file to change the first label entry to look as below:


```
label linux
  menu label ^Install Red Hat Enterprise Linux 7.3
  menu default
  kernel vmlinuz
  append initrd=initrd.img inst.stage2=hd:LABEL=RHEL-7.3\x20Server.x86_64
  inst.ks=hd:LABEL=Docker:ks.cfg quiet
```



Follow these steps to create boot iso image for the C-Series/ Second architecture:

1. RHEL7.3 base operating system distribution media does not include driver for raid controller (UCSC-RAID-M5) with C220 M5 servers. Copy megaraid_sas driver ISO file to /var/www/html/vMedia/isolinux directory
2. The driver ISO for C220 M5 megaraid_sas driver can be downloaded from Cisco.com: [https://software.cisco.com/download/release.html?mdfid=283862063&softwareid=283853158&release=3.1\(2\)&relind=AVAILABLE&rellifecycle=&reltype=latest](https://software.cisco.com/download/release.html?mdfid=283862063&softwareid=283853158&release=3.1(2)&relind=AVAILABLE&rellifecycle=&reltype=latest)
3. The required megaraid_sas driver ISO can be extracted by navigating to the path: ISO -> storage -> LSI -> UCS-RAID-M5 -> RHEL -> RHEL7.3 -> megaraid_sas-07.701.19.00_el7.3-1.x86_64.iso
4. Edit Isolinux/isolinux.cfg file for C-Series architecture as:


```
label linux
  menu label ^Install Red Hat Enterprise Linux 7.3
  menu default
  kernel vmlinuz
  append initrd=initrd.img inst.dd=path:/run/install/repo/megaraid_sas-07.701.19.00_el7.3-1.x86_64.iso inst.stage2=hd:LABEL=RHEL-7.3\x20Server.x86_64
  inst.ks=hd:LABEL=Docker:ks.cfg quiet
```
5. The required megaraid_sas driver ISO can be extracted by navigating to the path: ISO -> stor-


```
age -> LSI -> UCS-RAID-M5 -> RHEL -> RHEL7.3 -> megaraid_sas-07.701.19.00_el7.3-1.x86_64.iso
```



`menu default` is added, the text following `label check` is commented out, which is a default media check option. `inst.ks` makes the installation process automatic. `Docker` label attached to ks.cfg allows us to use a unique kickstart files for each of the cluster nodes enabling us to customize them.

5. Edit isolinux/isolinux.cfg file to change the timeout value on lower side, so that system starts the installation process without delay:

```
timeout 10
```

6. Build ISO image using following command:

```
# mkisofs -o redhat7.3-boot.iso -b isolinux.bin -c boot.cat -no-emul-boot -V 'RHEL-7.3
Server.x86_64' -boot-load-size 4 -boot-info-table -r -J -v -T isolinux/
<snip>
<snip>
Total translation table size: 6694
Total rockridge attributes bytes: 2601
Total directory bytes: 6144
Path table size(bytes): 54
Done with: The File(s)                                Block(s)      254532
Writing:      Ending Padblock                          Start Block 254571
Done with: Ending Padblock                             Block(s)      150
Max brk space used 1c000
254721 extents written (497 MB)
```

7. vMedia directory shows the boot ISO image ready. Copy this image to `install` directory, as this directory path is used in the vMedia policy created in UCS Manager:

```
vMedia]# ls -ltr
total 509444
dr-xr-xr-x. 4 root root      279 Dec  2 04:56 isolinux
-rw-r--r--. 1 root root 521668608 Dec  2 04:59 redhat7.3-boot.iso

# cp redhat7.3-boot.iso ../install/
```

Creating Kickstart Images

In order to mount kickstart configuration file for continuing the installation after boot images are loaded on to the bare metal cluster nodes, the HDD image should be provided which has the embedded ks.cfg.

1. Following is a sample kickstart file which is best suited for Docker EE cluster with minimal operating system requirements:

```
#version=DEVEL
# System authorization information
auth --enableshadow --passalgo=sha512
#repo --name="Server-HighAvailability" --
baseurl=file:///run/install/repo/addons/HighAvailability
#repo --name="Server-ResilientStorage" --
baseurl=file:///run/install/repo/addons/ResilientStorage
# Use CDROM installation media
#cdrom
# Use graphical install
graphical
# Run the Setup Agent on first boot
firstboot --disable
ignoredisk --only-use=sda
```

```

# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8

# Network information
install
url --url=http://<web-server ip>/ISO
#network --device eno6 --bootproto dhcp
#network --bootproto=dhcp --device=eno6 --onboot=off --ipv6=auto
network --device=eno5 --activate --bootproto=static --ip=<ip address> --
netmask=<netmask> --gateway=<gateway ip> --nameserver=<name server ip>
network --hostname=<hostname>

# Root password
rootpw --iscrypted <your encrypted root password>
# System services
services --disabled="chronyd"
# System timezone
timezone Asia/Kolkata --isUtc --nntp
user --name=cluster-admin --password=<your encrypted password> --iscrypted --
gecos="cluster-admin"
# X Window System configuration information
xconfig --startxonboot
# System bootloader configuration
bootloader --append="crashkernel=auto" --location=mbr --boot-drive=sda
autopart --type=lvm
# Partition clearing information
clearpart --none --initlabel

## reboot after the installation.
reboot

%packages
@base
@compat-libraries
@core
@desktop-debugging
@development
@dial-up
@file-server
@fonts
@gnome-desktop
@guest-agents
@guest-desktop-agents
@input-methods
@internet-browser
@java-platform
@multimedia
@network-file-system-client
@print-client
@x11
kexec-tools

%end

%post
echo "search cisco.com" > /etc/resolv.conf
echo "nameserver <name server ip>" >> /etc/resolv.conf
#---- Install our SSH key ----
mkdir -m0700 /root/.ssh/
cat <<EOF >/root/.ssh/authorized_keys
<< your ssh-rsa public key >>

```

```

EOF
### set permissions
chmod 0600 /root/.ssh/authorized_keys

addon com_redhat_kdump --enable --reserve-mb='auto'

%end

```



The above kickstart file needs user input values to be updated as per your environment. Following input values need to be updated -

1. Installation Media hosting url - url --url=http://<replace with your web host ip>/ISO
2. Host name, ip address, gateway, netmask and name servers should be replaced with applicable ones.
3. Network device name (eno5) should have remained same between the B-series servers and C-series servers from both the architectures, as long as there are no additional h/w installed and BIOS policy with Consistent Device Names (CDN) enablement have been used
4. For B-Series/first architecture under # System bootloader configuration, `--boot-drive=sda` should be used. On the other hand, for C-Series/second architecture this should be modified as `--boot-drive=sdd`. Also, for second architecture under # Run the Setup Agent on first boot, `ignoredisk --only-use=sda` should be changed to `sdd`. However, for first architecture, make sure to keep `ignoredisk --only-use=sda`.
5. Root password should be replaced with encrypted root password of your choice. To generate `openssl passwd -1` can be used to encrypt root password
6. Non-root user name and password should be changed also. Same method as given above can be used to generate encrypted password for the non-root user
7. In order to allow password-less ssh access to DEE cluster nodes from build/web-server, %post section should have copy pasted authorized_keys for root user. This will help in subsequent automated post-installation tasks using Ansible playbook

2. Following are the steps to generate .img file for a given ks.cfg:

```

# cp /root/ks.cfg /var/www/html/install/

# cd /var/www/html/install/

# fallocate -l 1M DEE-Ctrl-1.img
# ls -ltr
total 510472
-rw-r--r--. 1 root root 521668608 Dec  2 05:04 redhat7.3-boot.iso
-rwxr-xr-x. 1 root root      2594 Dec  2 05:48 ks.cfg
-rw-r--r--. 1 root root  1048576 Dec  2 05:49 DEE-Ctrl-1.img

# dd if=/dev/zero of=DEE-Ctrl-1.img bs=1M count=1 <<< where DEE-Ctrl-1 is the hostname

1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.001038 s, 1.0 GB/s

# mkfs -t ext4 DEE-Ctrl-1.img
mke2fs 1.42.9 (28-Dec-2013)
DEE-Ctrl-1.img is not a block special device.
Proceed anyway? (y,n) y

Filesystem too small for a journal
Discarding device blocks: done
Filesystem label=

```

```

OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
128 inodes, 1024 blocks
51 blocks (4.98%) reserved for the super user
First data block=1
Maximum filesystem blocks=1048576
1 block group
8192 blocks per group, 8192 fragments per group
128 inodes per group

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

# mkdir mnt

# mount -o loop DEE-Ctrl-1.img mnt/

# cp ks.cfg mnt/

# umount mnt/
# e2label DEE-Ctrl-1.img Docker <<<< This should match exactly with isolinux.cfg entry

# blkid DEE-Ctrl-1.img
DEE-Ctrl-1.img: LABEL="Docker" UUID="ee1fa4d2-2ae2-4318-8706-d5bc0ae0be62" TYPE="ext4"
<<< this verifies that image file labeled correctly.

```



The kickstart file name must match with the cluster node service profile name.

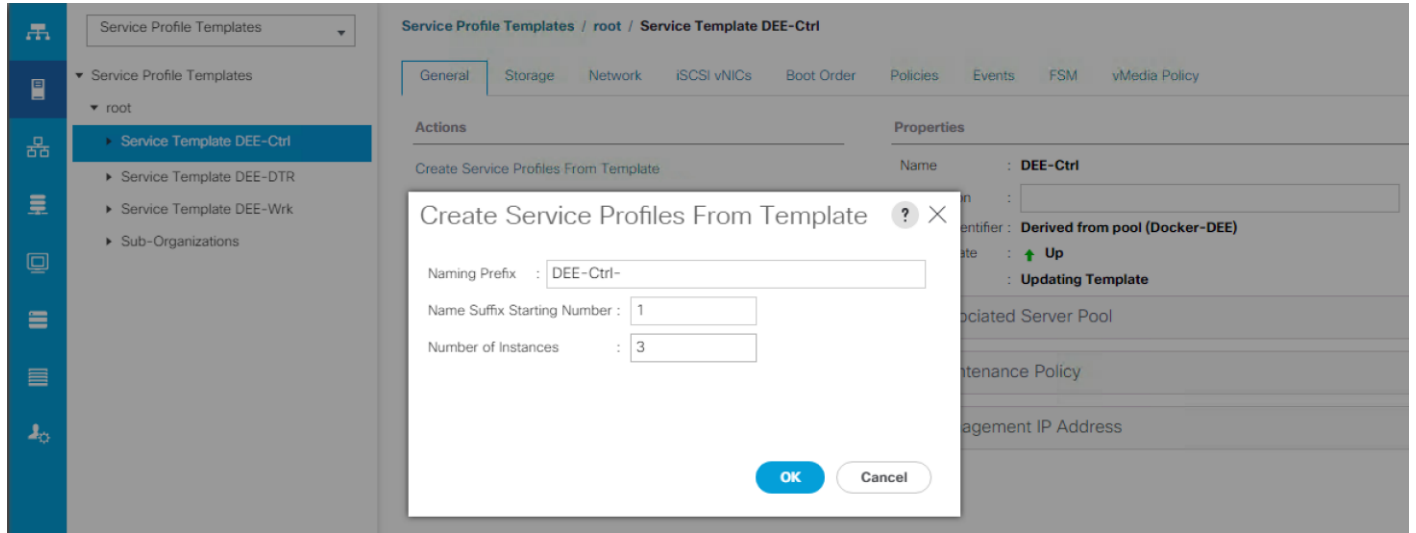
Service Profile Instantiation and Association

Service Profile Instantiation

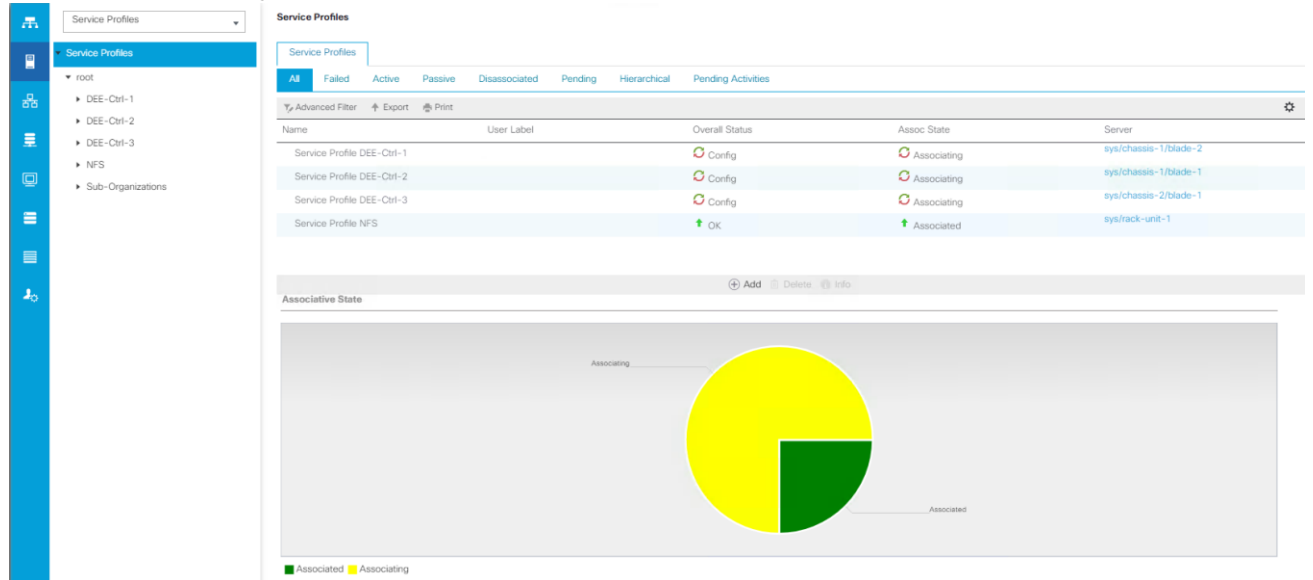
In these steps, service profiles are instantiated for DTR, UCP manager/controller and UCP worker nodes from their respective templates.

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

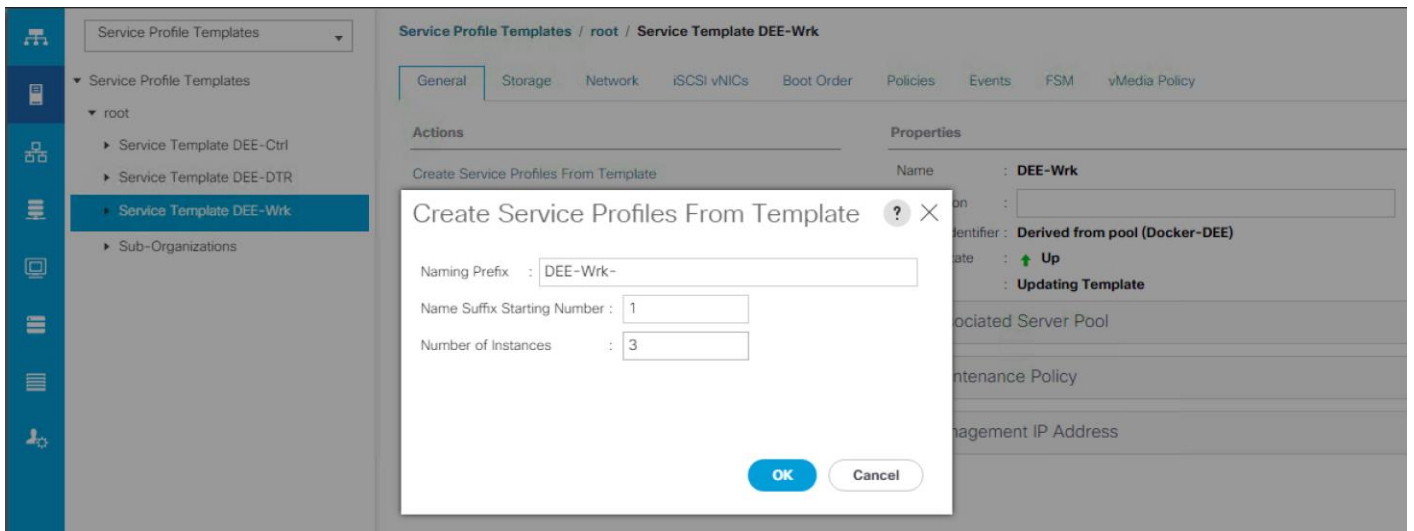
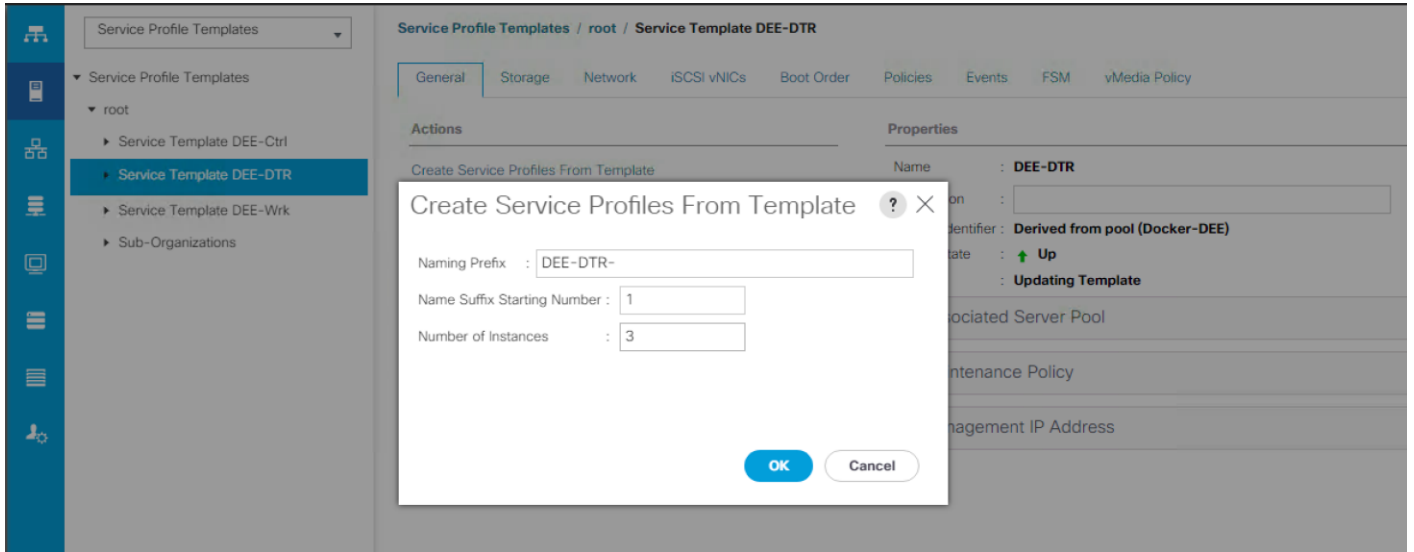
1. From Cisco UCS Manager, click Servers tab in the navigation pane.
3. Expand Servers > Service Profile Templates.
4. Right-click on the specific template (for example, DEE-Ctrl) and select Create Service Profiles from Template to open the Create Service Profile window.
5. In the Create Service Profile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



This will automatically associate service profiles to the servers, present in the pools.



6. Similarly instantiate other two service profiles (for example, DEE-DTR and DEE-Wrk nodes).



7. After the server pool association and service profile association gets completed, you can see all the services profiles shown in the associated state as shown below.

The screenshot shows the UCS Manager interface for Service Profiles. On the left is a navigation tree with 'Service Profiles' selected. The main area displays a table of service profiles and an 'Associative State' diagram.

Name	User Label	Overall Status	Assoc State	Server
Service Profile DEE-Ctrl-1		OK	Associated	sys/chassis-1/blade-6
Service Profile DEE-Ctrl-2		OK	Associated	sys/chassis-1/blade-1
Service Profile DEE-Ctrl-3		OK	Associated	sys/chassis-2/blade-1
Service Profile DEE-DTR-1		OK	Associated	sys/chassis-1/blade-3
Service Profile DEE-DTR-2		OK	Associated	sys/chassis-2/blade-3
Service Profile DEE-DTR-3		OK	Associated	sys/chassis-2/blade-2

The 'Associative State' diagram shows a large green circle representing the state of the profiles, with the word 'Associated' written next to it.



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

The screenshot shows the UCS Manager interface for Service Profiles. On the left is a navigation tree with 'Service Profiles' selected. The main area displays a table of service profiles and an 'Associative State' diagram.

Name	User Label	Overall Status	Assoc State	Server
Service Profile DEE-Ctrl-C-1		OK	Associated	sys/rack-unit-4
Service Profile DEE-Ctrl-C-2		OK	Associated	sys/rack-unit-3
Service Profile DEE-Ctrl-C-3		OK	Associated	sys/rack-unit-2
Service Profile DEE-Wrk-C-1		OK	Associated	sys/rack-unit-1

The 'Associative State' diagram shows a large green circle representing the state of the profiles, with the word 'Associated' written next to it.

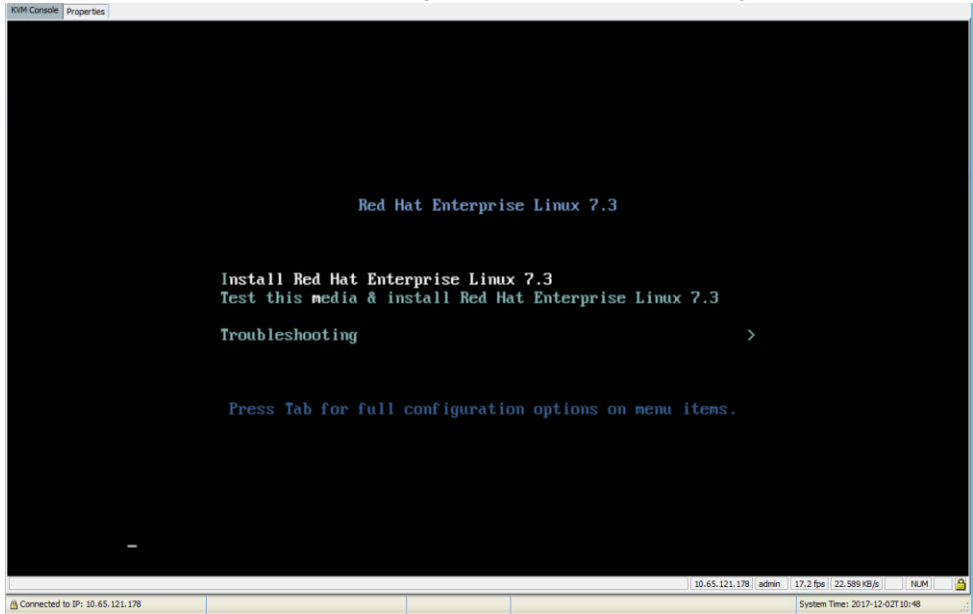
Installation of Red Hat Enterprise Linux Operating System

UCS Manager's vMedia policy feature is used in this solution to enable automated bare metal OS installation. The installation process gets initialized automatically once the service profiles are associated.

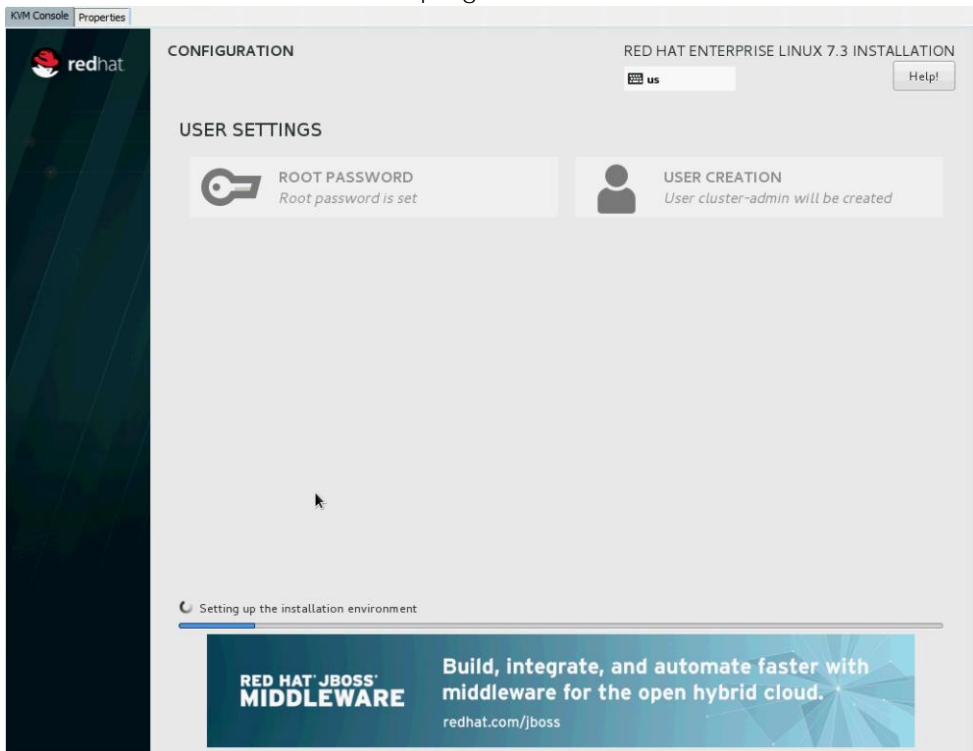
1. vMedia policy at UCS Manager for CIMC mounted CD/DVD and HDD image files, needed for bare metal install
2. A web server hosting Boot ISO, Kickstart HDD image and rest of the installation media
3. Boot policy having vMedia devices to be available for bootstrapping and subsequent operating system install

Following steps shows the automated OS install workflow in various stages:

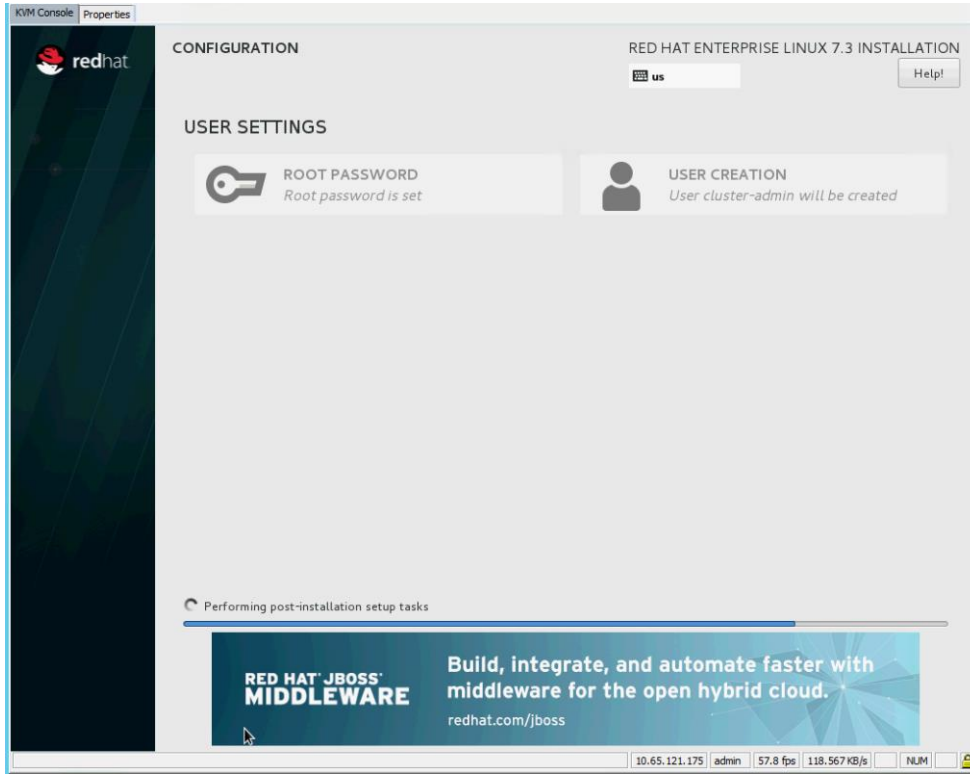
1. Once the service profiles get associated, boot ISO gets picked up for booting kernel



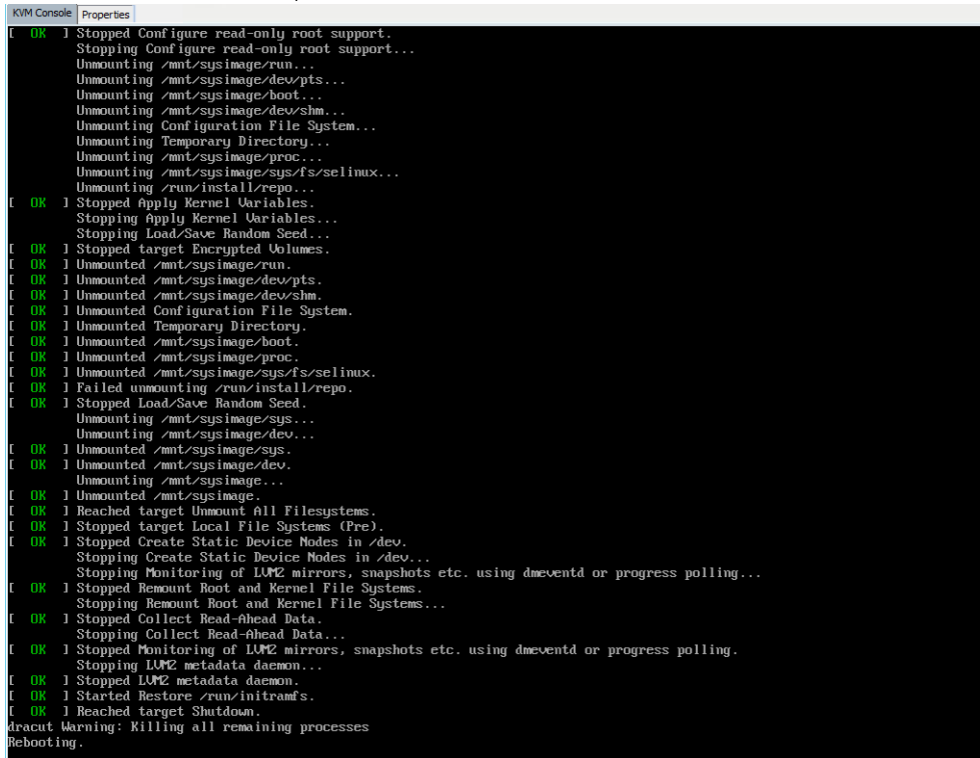
2. RHEL 7.3 installation is in progress



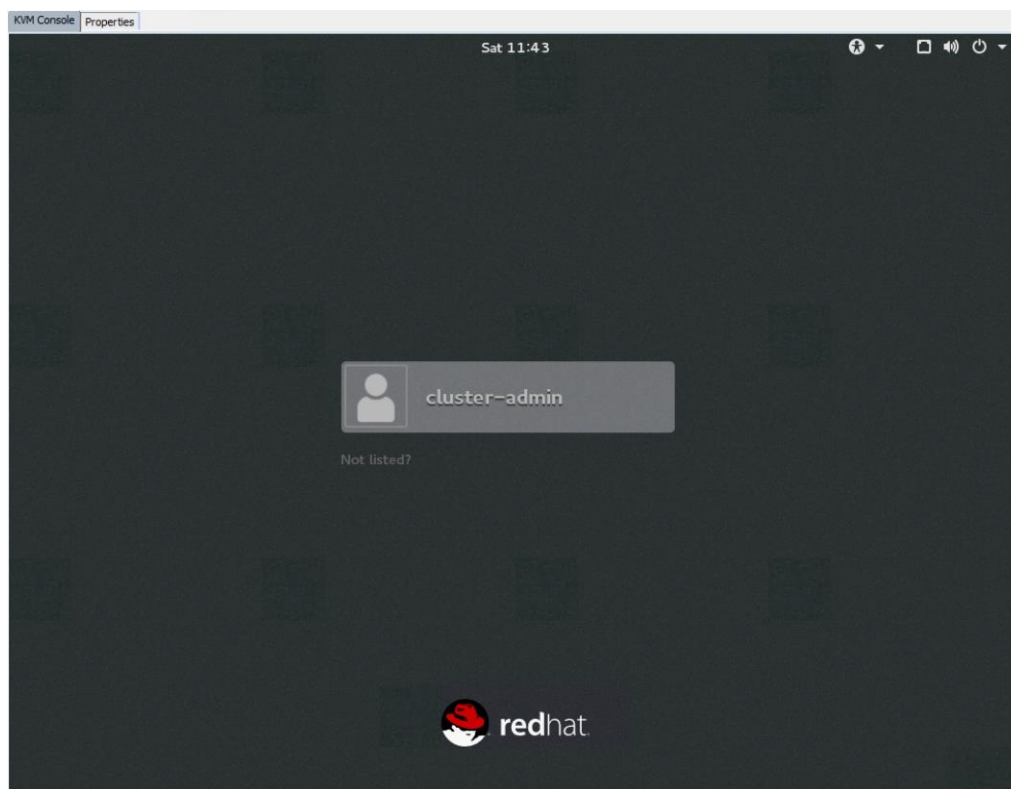
3. Post installation tasks progress indication



4. Server reboots post OS installation



5. Host OS is up and running



At the UCS Manager level, CIMC mounted boot ISO and kickstart HDD image gets mounted at the boot time on the host as shown below, when it boots up for the first time. Need Cisco UCS Manager CLI access to monitor the boot process:

```
Docker-A# scope server 1/1
Docker-A /chassis/server # scope cimc
Docker-A /chassis/server/cimc # show vmedia-mapping-list expand detail
```

```
Vmedia Mapping List:
  Full Name: sys/chassis-1/blade-1/mgmt/actual-mount-list
```

```
Vmedia Mapping:
  Vdisk Id: 1
  Mapping Name: RHEL7
  Device Type: Cdd
  Remote IP: 10.65.122.80
  Image Path: install
  Image File Name: redhat7.3-boot.iso
  Mount Protocol: Http
  Port Number: 80
  Mount Status: Mounted
  Error: None
  Password:
  User ID:
  Authentication Protocol: None
  Remap Mount on Host OS Eject: No

  Vdisk Id: 2
  Mapping Name: ksimage
  Device Type: Hdd
  Remote IP: 10.65.122.80
  Image Path: install
  Image File Name: DEE-Ctrl-2.img
```

```

Mount Protocol: Http
Port Number: 80
Mount Status: Mounted
Error: None
Password:
User ID:
Authentication Protocol: None
Remap Mount on Host OS Eject: No

```



Installation workflow remains the same for both the architectures.

Docker Enterprise Edition Installation

This section provides detailed instructions on installing Docker Enterprise Edition cluster nodes. Ansible playbook holds a comprehensive list of post OS installation tasks. On running the Ansible playbook post OS install tasks get executed step by step to get the Docker EE up and running on all the cluster nodes along with their defined node roles. Following steps are executed on running the Ansible playbook:

1. Post installation configurations, which include:
 - a. Setting up environment, example: proxy settings
 - b. Updating Cisco VIC enic driver to the latest async driver compatible with UCS Manager release
 - c. Registering system to Red Hat Subscription Management, attaching to rhel-7-server repo and completing Yum update
 - d. Configuring NTP and firewall required for Docker EE cluster
 - e. Configuring Storage for Docker **EE. Recommended for production use case 'Device Mapper Driver in Direct LVM mode'**
 - f. Installing Docker EE engine
 - g. Installing Docker UCP – Manager, Manager replicas, workers, DTR and DTR replicas
2. Setting up Ansible on a build server which is used for automated OS install
3. Setting up Ansible playbook for configuring all the above mentioned tasks required for Docker EE

Configuring Firewall Ports for Docker EE

Docker Enterprise Edition 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 EE Engine and the Docker UCP. For opening ports on hosts, a separate task has been created in the Ansible playbook. For every port type such as TCP and UDP, refer the table below to see the specific ports to be opened with details.

Table 11 TCP and UDP ports to be opened on hosts for Docker UCP

Hosts	Direction	Port	Purpose
managers, workers	in	TCP 443 (configurable)	Web app and CLI client access to UCP

managers	in	TCP 2376 (configurable)	Port for the Docker Swarm manager. Used for backwards compatibility
managers, workers	In, out	TCP 2377	Port for communication between Swarm nodes
managers, workers	in, out	UDP 4789	Overlay networking
managers, workers	in, out	TCP + UDP 7946	Port for Gossip-based clustering
managers, workers	in	TCP 12376	Proxy for TLS, provides access to UCP, Swarm, and Engine
managers	in	TCP 12379	Internal node configuration, cluster configuration, and HA
managers	in	TCP 12380	Internal node configuration, cluster configuration, and HA
managers	in	TCP 12381	Port for certificate authority
managers	in	TCP 12382	Port for UCP certificate authority
managers	in	TCP 12383	Used by the authentication storage backend
managers	in	TCP 12384	Used by authentication storage backend for replication across controllers
managers	in	TCP 12385	The port where the authentication API is exposed
managers	in	TCP 12386	Used by the authentication worker

Ansible Installation

Ansible is an open source automation tool for configuration management and provisioning. Drudgery tasks such as post installation tasks can be quickly achieved using Ansible. This powerful tool needs a very few steps to configure a large cluster nodes to get them **up and running**. **This tool is not based on a 'server-client' model for executing automated tasks. Ansible tool can be installed through rhel-7-server-extra-rpms repo.**

Ansible controller node does not require any additional software or packages. It's the node from where Ansible commands/playbook is executed for automated configuration of all the nodes including the controller nodes itself. Note that the Ansible controller node can be part of Docker EE cluster nodes itself.



This solution uses a build/web-server node for automated PXE-less OS installation. With a separate build node yum update is run on all the cluster nodes at once and all the nodes are rebooted after the update is complete. Once the nodes have rebooted, the Ansible playbook tasks will resume from the point where it had stopped to reboot the nodes.

You can choose to not use a build/web-server and use one of the cluster nodes itself as an Ansible controller node and execute the playbook from that node. But with this, you need to do a 'yum update' manually on all the cluster nodes separately.

Following are the steps to install and configure Ansible:

1. Attach extra-rpm repo on the node from where you plan to run the playbook

```
# yum-config-manager --enable rhel-7-server-extras-rpms

# yum info ansible
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
Available Packages
Name       : ansible
Arch      : noarch
Version   : 2.4.1.0
Release   : 1.e17
Size      : 7.6 M
Repo      : rhel-7-server-extras-rpms/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.

# yum install Ansible
<snip>
<snip>
Installed:
  ansible.noarch 0:2.4.1.0-1.e17

Dependency Installed:
  python-babel.noarch 0:0.9.6-8.e17          python-cffi.x86_64 0:1.6.0-5.e17
  python-enum34.noarch 0:1.0.4-1.e17         python-httplib2.noarch 0:0.9.2-1.e17
  python-idna.noarch 0:2.0-1.e17           python-ipaddress.noarch 0:1.0.16-2.e17
  python-jinja2.noarch 0:2.7.2-2.e17       python-markupsafe.x86_64 0:0.11-10.e17
  python-paramiko.noarch 0:2.1.1-2.e17     python-passlib.noarch 0:1.6.5-2.e17
  python-ply.noarch 0:3.4-10.e17          python-pycparser.noarch 0:2.14-1.e17
```

```
python2-cryptography.x86_64 0:1.3.1-3.el7 python2-jmespath.noarch 0:0.9.0-3.el7
python2-pyasnl.noarch 0:0.1.9-7.el7 sshpass.x86_64 0:1.06-2.el7
```

Complete!

2. Generate and populate ssh key on the Ansible controller node (build/web-server) to 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.

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in y.
Your public key has been saved in y.pub.
The key fingerprint is:
81:7a:0b:35:81:76:1b:fe:76:d8:9d:e3:0d:7a:89:a8 root@DEE-Wrk-C-1.cisco.com
The key's randomart image is:
+--[ RSA 2048]-----+
|      ..          |
|     o oo        |
|    . o+o.       |
|     oo. .       |
|    o ..So . .   |
|     o .+ o =    |
|      .. o + =   |
|       . o + .   |
|        E.      .|
+-----+

```



SSH key is generated on the build/web-server and is included in kickstart file ks.cfg. In this way the key got distributed on all the cluster nodes as part of OS install itself.

3. Edit /etc/ansible/hosts file on the build/web-server, to include hostnames of DEE cluster nodes at the end of the file

```
[DEE-Nodes]
DEE-Ctrl-1
DEE-Ctrl-2
DEE-Ctrl-3
DEE-DTR-1
DEE-DTR-2
DEE-DTR-3
DEE-Wrk-1
DEE-Wrk-2
DEE-Wrk-3
DEE-Wrk-4
```

4. Verify you can login to cluster-nodes with password-less ssh access

```
# ansible DEE-Nodes -m shell -a "uptime"

DEE-Ctrl-2 | SUCCESS | rc=0 >>
 17:33:52 up 5:51, 2 users, load average: 0.01, 0.04, 0.05

DEE-Ctrl-1 | SUCCESS | rc=0 >>
 17:34:18 up 2:48, 2 users, load average: 0.13, 0.09, 0.06

DEE-Ctrl-3 | SUCCESS | rc=0 >>
 17:34:10 up 5:50, 2 users, load average: 0.23, 0.11, 0.07

DEE-DTR-1 | SUCCESS | rc=0 >>
 17:34:02 up 2:49, 2 users, load average: 0.06, 0.06, 0.05
```

```

DEE-DTR-2 | SUCCESS | rc=0 >>
 17:34:04 up  2:47,  2 users,  load average: 0.00, 0.01, 0.05

DEE-Wrk-1 | SUCCESS | rc=0 >>
 17:34:01 up  2:02,  2 users,  load average: 0.03, 0.05, 0.05

DEE-Wrk-3 | SUCCESS | rc=0 >>
 17:34:05 up  2:03,  2 users,  load average: 0.04, 0.10, 0.08

DEE-DTR-3 | SUCCESS | rc=0 >>
 17:35:10 up  2:47,  2 users,  load average: 0.12, 0.15, 0.11

DEE-Wrk-4 | SUCCESS | rc=0 >>
 17:34:02 up  2:48,  2 users,  load average: 0.09, 0.11, 0.13

DEE-Wrk-2 | SUCCESS | rc=0 >>
 17:34:14 up  2:47,  2 users,  load average: 0.27, 0.14, 0.13

```

Ansible Playbook Execution

To run the playbook, you need to download the entire directory structure on a build node and run the playbook command, after editing DEE-Nodes/DEE-C-Nodes files with cluster node details, enter the variables in the group_vars/all file.

Configuration details on DEE-Nodes/DEE-C-Nodes along with the YAML file used for executing the Ansible playbook are shown in the Appendix section.



For first architecture group_var/all variables would be:

```

ntp_server: "<your NTP server IP>"
http_proxy: http://<Proxy IP>:<port>/
https_proxy: https:// ://<Proxy IP>:<port>/
http_proxy_hostname: <Proxy host IP>
proxy_port: <proxy port>
node01: <node01 IP>
node01_name: <node01_name>
node01_fqdn: <FQDN node01 name>
rhsm_user: "<user_name>"
rhsm_password: "<password>"
pool_id: "<your entitled pool-id>"
UCP_Manager: "<your designated first Manager Node IP>"
UCP_Admin: "admin"
UCP_Admin_Pass: "<your choosen password>"
DTR_NFS_URL: "nfs://<NFS Host IP>/DTR-NFS"
UCP_URL: "https://<your designated first Manager Node IP>:443"
UCP_Port: "443"
UCP_Ver: "2.2.4"
DTR_Ver: "2.4.0"

```



For second architecture group_var/all variables would be, same except for below. As there will co-hosting for UCP/DTR services on the common nodes:

```

UCP_URL: "https://<your designated first Manager Node IP>:4443"
UCP_Port: "4443"

```

```

# ansible-playbook -verbose -i /etc/ansible/DEE-Nodes DEE-Nodes.yml -u root
Using /etc/ansible/ansible.cfg as config file

```

```

PLAY [DEE-Nodes]
*****
<snip>
<snip>
TASK [common : Copying rhsm conf]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "checksum":
"8823ab9f7a5968660aaadfb5b2a7c3bec41a4574", "dest": "/etc/rhsm/rhsm.conf", "gid": 0,
"group": "root", "md5sum": "9fdb176428244189b6cb3a509474529e", "mode": "0644", "owner":
"root", "secontext": "system_u:object_r:etc_t:s0", "size": 1680, "src":
"/root/.ansible/tmp/ansible-tmp-1512223789.57-58261853961485/source", "state": "file",
"uid": 0}
<snip>
<snip>
TASK [common : Copying NTP conf]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "checksum":
"8b0ef00a20b0de2714dd3f5784f0c942a5ffc218", "dest": "/etc/ntp.conf", "gid": 0, "group":
"root", "md5sum": "2ac3062f4c954f63d5caflb707ce5fcf", "mode": "0644", "owner": "root",
"secontext": "system_u:object_r:net_conf_t:s0", "size": 2187, "src":
"/root/.ansible/tmp/ansible-tmp-1512223790.86-29764737763044/source", "state": "file",
"uid": 0}
<snip>
<snip>
TASK [common : Copying enic driver]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "checksum":
"a5036b36c083492b28fd04e8ac884a5417f5fc71", "dest": "/root/kmod-enic-2.3.0.44-
rhel7u3.el7.x86_64.rpm", "gid": 0, "group": "root", "md5sum":
"e86fd8cb351ae5bdb4781aa0eabbbea7", "mode": "0644", "owner": "root", "secontext":
"system_u:object_r:admin_home_t:s0", "size": 770700, "src": "/root/.ansible/tmp/ansible-
tmp-1512223793.39-31250091169693/source", "state": "file", "uid": 0}
<snip>
<snip>
TASK [yum : Registering System to RHSM]
*****
changed: [DEE-Ctrl-3] => {"changed": true, "cmd": "subscription-manager register --
username=cisco_rkharya --password=qwerty123", "delta": "0:00:21.886475", "end": "2017-12-
02 19:41:49.373390", "rc": 0, "start": "2017-12-02 19:41:27.486915", "stderr": "",
"stderr_lines": [], "stdout": "Registering to:
subscription.rhn.redhat.com:443/subscription\nThe system has been registered with ID:
80a3e8f4-46ed-4681-ba24-5d863a5fc285 ", "stdout_lines": ["Registering to:
subscription.rhn.redhat.com:443/subscription", "The system has been registered with ID:
80a3e8f4-46ed-4681-ba24-5d863a5fc285 "]}
<snip>
<snip>
TASK [yum : Refresh Subscription Manager DB]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "subscription-manager refresh", "delta":
"0:00:06.599053", "end": "2017-12-02 19:42:26.407339", "rc": 0, "start": "2017-12-02
19:42:19.808286", "stderr": "", "stderr_lines": [], "stdout": "All local data refreshed",
"stdout_lines": ["All local data refreshed"]}
<snip>
<snip>
TASK [yum : Subscribing to Pool]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "subscription-manager attach --
pool=8a85f9815ab5216e015ab56c80c3636c", "delta": "0:00:20.794736", "end": "2017-12-02
19:42:55.457214", "rc": 0, "start": "2017-12-02 19:42:34.662478", "stderr": "",
"stderr_lines": [], "stdout": "Successfully attached a subscription for: Red Hat
Enterprise Linux Server with Smart Management, Premium (Physical or Virtual Nodes)",

```



```

"stdout_lines": ["Successfully attached a subscription for: Red Hat Enterprise Linux
Server with Smart Management, Premium (Physical or Virtual Nodes)"]
<snip>
<snip>
TASK [yum : Setting RHEL release version]
*****
changed: [DEE-Ctrl-3] => {"changed": true, "cmd": "subscription-manager release --
set=7.3", "delta": "0:00:06.754070", "end": "2017-12-02 19:43:18.344113", "rc": 0,
"start": "2017-12-02 19:43:11.590043", "stderr": "", "stderr_lines": [], "stdout":
"Release set to: 7.3", "stdout_lines": ["Release set to: 7.3"]}
<snip>
<snip>
TASK [yum : Enabling Repos]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "subscription-manager repos --
enable=rhel-7-server-rpms", "delta": "0:00:22.388693", "end": "2017-12-02
19:43:55.139522", "rc": 0, "start": "2017-12-02 19:43:32.750829", "stderr": "",
"stderr_lines": [], "stdout": "Repository 'rhel-7-server-rpms' is enabled for this
system.", "stdout_lines": ["Repository 'rhel-7-server-rpms' is enabled for this system."]}
<snip>
<snip>
TASK [yum : install enic rpm from a local file]
*****
changed: [DEE-DTR-2] => {"changed": true, "msg": "", "rc": 0, "results": ["Loaded plugins:
langpacks, product-id, search-disabled-repos, subscription-\n
:
manager\nExamining /root/kmod-enic-2.3.0.44-rhel7u3.el7.x86_64.rpm: kmod-enic-2.3.0.44-
rhel7u3.el7.x86_64\nMarking /root/kmod-enic-2.3.0.44-rhel7u3.el7.x86_64.rpm to be
installed\nResolving Dependencies\n--> Running transaction check\n--> Package kmod-
enic.x86_64 0:2.3.0.44-rhel7u3.el7 will be installed\n--> Finished Dependency
Resolution\n\nDependencies
Resolved\n\n=====
==\n Package\n      Arch   Version                Repository
Size\n=====
stalling:\n kmod-enic\n      x86_64 2.3.0.44-rhel7u3.el7 /kmod-enic-2.3.0.44-
rhel7u3.el7.x86_64 4.3 M\n\nTransaction
Summary\n=====
nInstall 1 Package\n\nTotal size: 4.3 M\nInstalled size: 4.3 M\nDownloading
packages:\nRunning transaction check\nRunning transaction test\nTransaction test
succeeded\nRunning transaction\n Installing : kmod-enic-2.3.0.44-rhel7u3.el7.x86_64
1/1 \n/sbin/dracut: line 649: warning: setlocale: LC_CTYPE: cannot change locale (): No
such file or directory\n/sbin/dracut: line 649: warning: setlocale: LC_CTYPE: cannot
change locale (): No such file or directory\n Verifying : kmod-enic-2.3.0.44-
rhel7u3.el7.x86_64
1/1 \n\nInstalled:\n kmod-enic.x86_64
0:2.3.0.44-rhel7u3.el7
\n\nComplete!\n"]}]
<snip>
<snip>
TASK [yum : Yum Update]
*****
<snip>
<snip>
TASK [yum : Check for reboot hint]
*****
changed: [DEE-Ctrl-2] => {"changed": true, "cmd": "LAST_KERNEL=$(rpm -q --last kernel |
perl -pe 's/^kernel-(\\S+).*/$1/' | head -1);\n CURRENT_KERNEL=$(uname -r);\n if [
$LAST_KERNEL != $CURRENT_KERNEL ];\n then\n echo 'reboot'; else echo 'no';\n fi\n exit 0",
"delta": "0:00:00.113086", "end": "2017-12-02 20:41:05.089307", "rc": 0, "start": "2017-
12-02 20:41:04.976221", "stderr": "", "stderr_lines": [], "stdout": "reboot",
"stdout_lines": ["reboot"]}
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "LAST_KERNEL=$(rpm -q --last kernel |
perl -pe 's/^kernel-(\\S+).*/$1/' | head -1);\n CURRENT_KERNEL=$(uname -r);\n if [
$LAST_KERNEL != $CURRENT_KERNEL ];\n then\n echo 'reboot'; else echo 'no';\n fi\n exit 0",
"delta": "0:00:00.127061", "end": "2017-12-02 20:41:30.301489", "rc": 0, "start": "2017-

```

```

12-02 20:41:30.174428", "stderr": "", "stderr_lines": [], "stdout": "reboot",
"stdout_lines": ["reboot"]}
<snip>
<snip>
TASK [yum : Rebooting ...]
*****
changed: [DEE-Ctrl-1] => {"ansible_job_id": "583666472427.28160", "changed": true, "fin-
ished": 0, "results_file": "/root/.ansible_async/583666472427.28160", "started": 1}
changed: [DEE-Ctrl-2] => {"ansible_job_id": "105260474277.6348", "changed": true, "fin-
ished": 0, "results_file": "/root/.ansible_async/105260474277.6348", "started": 1}
<snip>
<snip>
TASK [yum : Wait for host to boot]
*****
ok: [DEE-DTR-1 -> localhost] => {"changed": false, "elapsed": 360, "path": null, "port":
22, "search_regex": null, "state": "started"}
ok: [DEE-Ctrl-1 -> localhost] => {"changed": false, "elapsed": 360, "path": null, "port":
22, "search_regex": null, "state": "started"}
<snip>
<snip>
TASK [storage : Device Mapper Driver Configuration - Logical Volume Creation]
*****
changed: [DEE-DTR-1] => {"changed": true, "cmd": "lvcreate --wipesignatures y -n thin-
poolmeta Docker -l 1%VG", "delta": "0:00:00.087190", "end": "2017-12-02 20:55:24.414561",
"rc": 0, "start": "2017-12-02 20:55:24.327371", "stderr": "", "stderr_lines": [],
"stdout": " Logical volume \"thinpoolmeta\" created.", "stdout_lines": [" Logical volume
\"thinpoolmeta\" created."]}
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "lvcreate --wipesignatures y -n thin-
poolmeta Docker -l 1%VG", "delta": "0:00:00.100413", "end": "2017-12-02 20:55:24.426827",
"rc": 0, "start": "2017-12-02 20:55:24.326414", "stderr": "", "stderr_lines": [],
"stdout": " Logical volume \"thinpoolmeta\" created.", "stdout_lines": [" Logical volume
\"thinpoolmeta\" created."]}
<snip>
<snip>
TASK [storage : Device Mapper Driver Configuration - Thinpool conversion]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "lvconvert -y --zero n -c 512K --
thinpool Docker/thinpool --poolmetadata Docker/thinpoolmeta", "delta": "0:00:00.244574",
"end": "2017-12-02 20:55:25.307003", "rc": 0, "start": "2017-12-02 20:55:25.062429",
"stderr": " WARNING: Converting logical volume Docker/thinpool and Docker/thinpoolmeta to
thin pool's data and metadata volumes with metadata wiping.\n THIS WILL DESTROY CONTENT
OF LOGICAL VOLUME (filesystem etc.)", "stderr_lines": [" WARNING: Converting logical vol-
ume Docker/thinpool and Docker/thinpoolmeta to thin pool's data and metadata volumes with
metadata wiping.", " THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)"],
"stdout": " Converted Docker/thinpool to thin pool.", "stdout_lines": [" Converted Dock-
er/thinpool to thin pool."]}
<snip>
<snip>
TASK [docker : Install Docker Enterprise Edition]
*****
changed: [DEE-Ctrl-3] => {"changed": true, "msg": "warning:
/var/cache/yum/x86_64/7Server/docker-ee-stable-17.06/packages/docker-ee-17.06.2.ee.6-
3.el7.rhel.x86_64.rpm: Header V4 RSA/SHA512 Signature, key ID 76682bc9: NOKEY\nImporting
GPG key 0x76682BC9:\n Userid      : \"Docker Release (EE rpm) <docker@docker.com>\"\n Fin-
gerprint: 77fe da13 1a83 1d29 a418 d3e8 99e5 ff2e 7668 2bc9\n From          :
https://storebits.docker.com/ee/linux/sub-3a55d8be-ded8-497f-97ca-
ee3289a7cdcd/rhel/gpg\n", "rc": 0, "results": ["Loaded plugins: langpacks, product-id,
search-disabled-repos, subscription-\n
: manager\nResolving Dependencies\n-->
Running transaction check\n--> Package docker-ee.x86_64 0:17.06.2.ee.6-3.el7.rhel will be
installed\n--> Finished Dependency Resolution\n\nDependencies Re-
solved\n\n=====
\n Package      Arch      Version                               Repository
Size\n=====
\nIn

```

```

stalling:\n docker-ee      x86_64      17.06.2.ee.6-3.el7.rhel      docker-ee-stable-17.06      25
M\n\nTransaction Sum-
mary\n=====
In
stall 1 Package\n\nTotal download size: 25 M\nInstalled size: 79 M\nDownloading packag-
es:\nPublic key for docker-ee-17.06.2.ee.6-3.el7.rhel.x86_64.rpm is not in-
stalled\nRetrieving key from https://storebits.docker.com/ee/linux/sub-3a55d8be-ded8-497f-
97ca-ee3289a7cdcd/rhel/gpg\nRunning transaction check\nRunning transaction
test\nTransaction test succeeded\nRunning transaction\n Installing : docker-ee-
17.06.2.ee.6-3.el7.rhel.x86_64      1/1 \n Verifying : docker-ee-
17.06.2.ee.6-3.el7.rhel.x86_64      1/1 \n\nInstalled:\n docker-ee.x86_64
0:17.06.2.ee.6-3.el7.rhel      \n\nComplete!\n"]}]
<snip>
<snip>
PLAY [UCP-Mgr]
*****
TASK [UCPswarm : copy the ucp license to the remote machine]
*****
changed: [DEE-Ctrl-1] => {"changed": true, "checksum":
"74ce90955cbc50e9d9b5bd84fd50e2e83495d0d6", "dest": "/tmp/docker_subscription.lic", "gid":
0, "group": "root", "md5sum": "1a39823db1747ce6ce18be8e4241deca", "mode": "0644", "owner":
"root", "secontext": "unconfined_u:object_r:admin_home_t:s0", "size": 3013, "src":
"/root/.ansible/tmp/ansible-tmp-1512228739.15-240326800829239/source", "state": "file",
"uid": 0}

TASK [UCPswarm : download and install ucp images]
*****
<snip>
<snip>
PLAY [UCP-DTR]
*****
TASK [UCPdtr : download and install DTR]
*****
TASK [UCPdtr : capture DTR replica ID]
*****
changed: [DEE-DTR-1] => {"changed": true, "cmd": "docker ps|grep dtr-nginx|awk '{print
$NF}' | cut -d'-' -f3", "delta": "0:00:00.023095", "end": "2017-12-02 22:09:02.729974",
"rc": 0, "start": "2017-12-02 22:09:02.706879", "stderr": "", "stderr_lines": [],
"stdout": "3092b164a12a", "stdout_lines": ["3092b164a12a"]}

PLAY [UCP-DTR-R1]
*****
***
TASK [UCPdtr-r1 : Adding Manager Replicas] *****

```

```

PLAY RECAP *****
DEE-Ctrl-1      : ok=68  changed=60  unreachable=0  failed=0
DEE-Ctrl-2      : ok=65  changed=57  unreachable=0  failed=0
DEE-Ctrl-3      : ok=65  changed=57  unreachable=0  failed=0
DEE-DTR-1       : ok=67  changed=59  unreachable=0  failed=0
DEE-DTR-2       : ok=66  changed=58  unreachable=0  failed=0
DEE-DTR-3       : ok=66  changed=58  unreachable=0  failed=0
DEE-Wrk-1       : ok=65  changed=57  unreachable=0  failed=0
DEE-Wrk-2       : ok=65  changed=57  unreachable=0  failed=0
DEE-Wrk-3       : ok=65  changed=57  unreachable=0  failed=0
DEE-Wrk-4       : ok=65  changed=57  unreachable=0  failed=0

```

The PLAY-RECAP screen shot above shows no failures, indicating that the installation was successful. After this, Verify Docker UCP and DTR - UI dashboards and CLI interfaces are up and running.

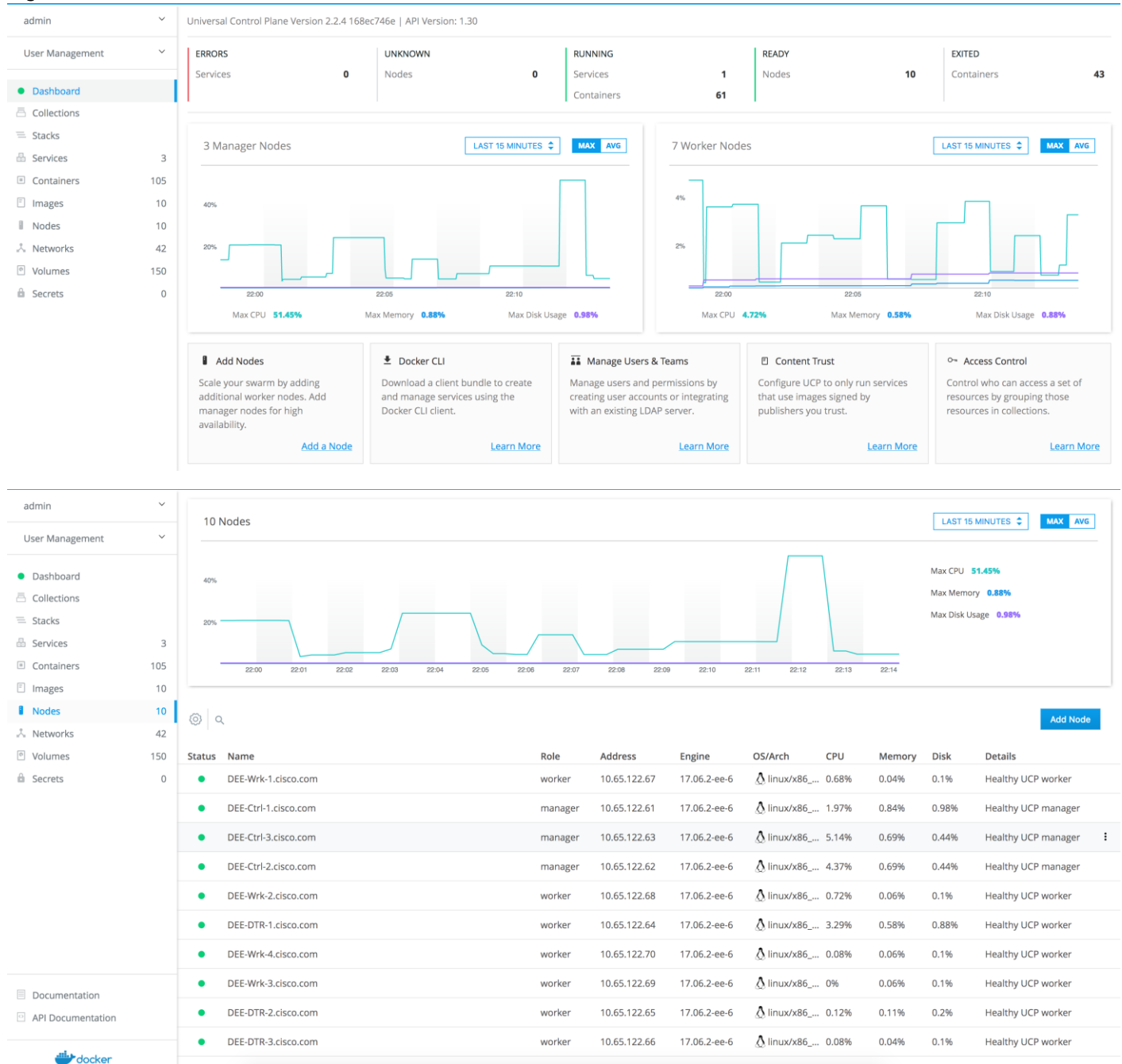
Verifying Docker Enterprise Edition Installation

This section shows the validation of the entire Docker EE through Ansible playbook.

Docker UCP UI

The following figure shows the UCP dashboard with all the 10 nodes seen as healthy.

Figure 26 Docker UCP dashboard



DTR UI

The following figures show the DTR UI.

Figure 27 DTR landing page

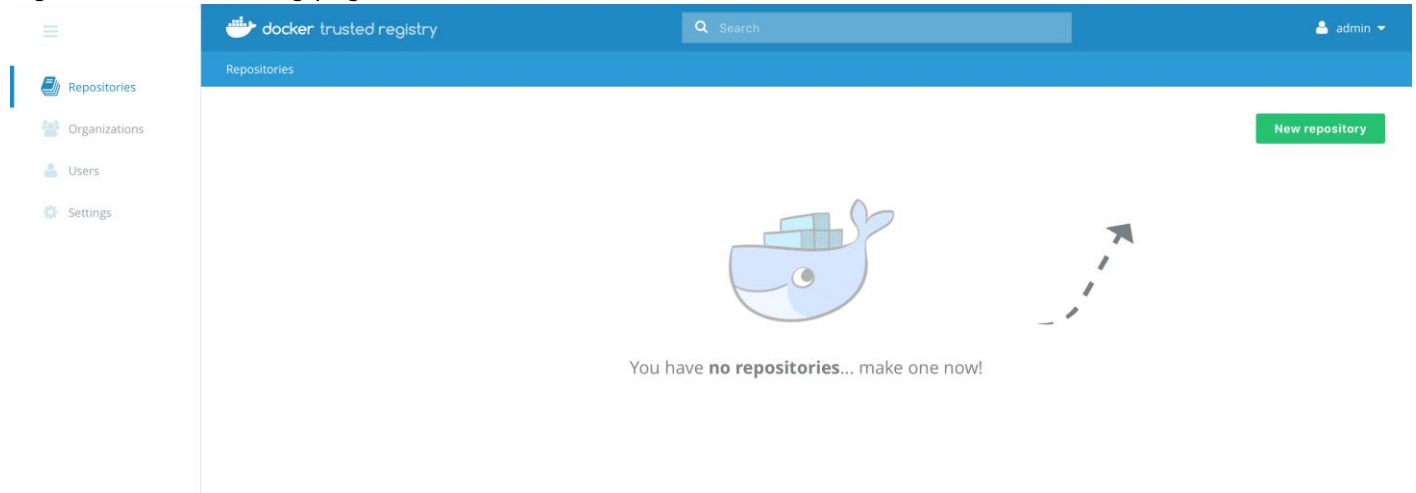
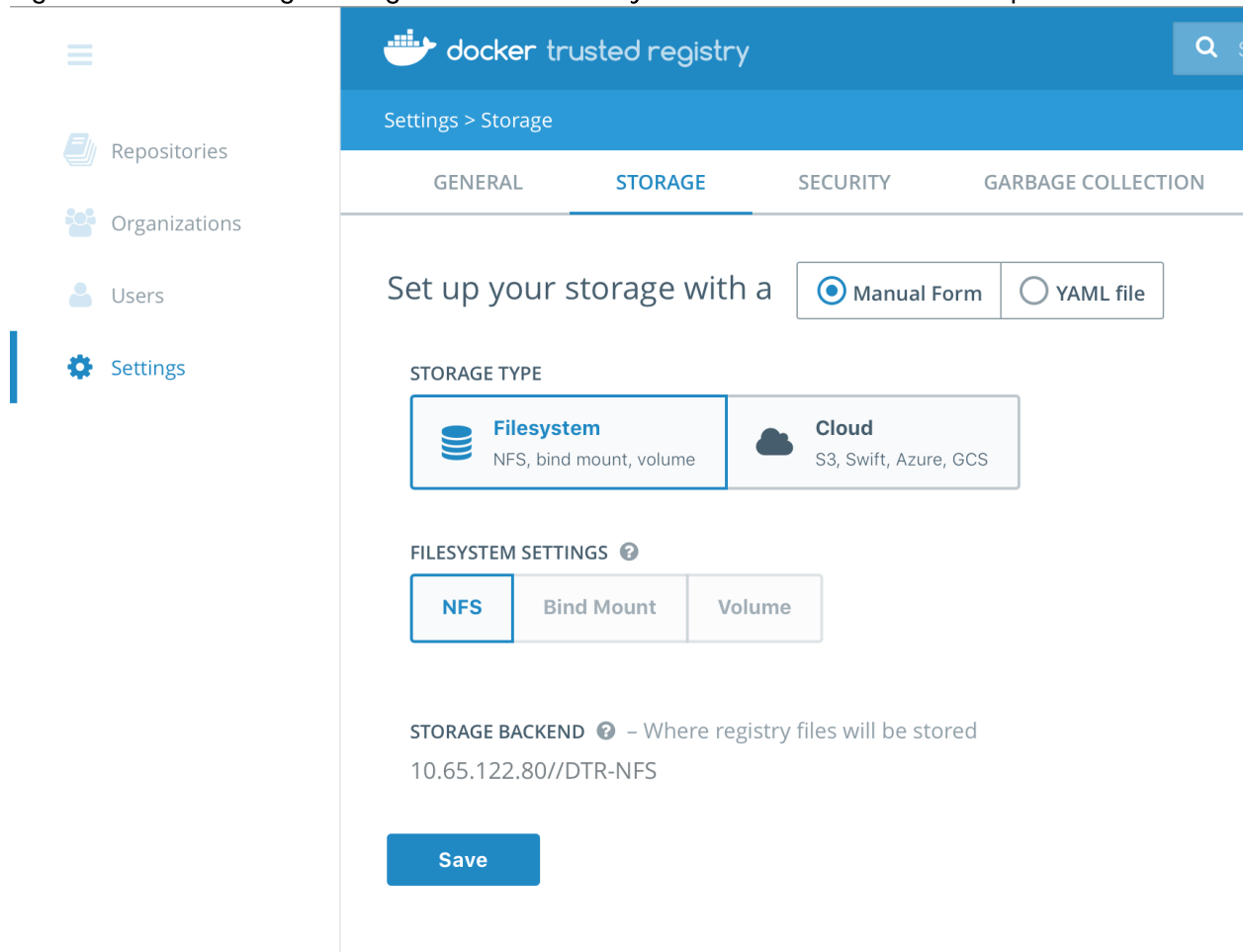


Figure 28 DTR Storage Configuration: NFS file-system shared across 3 DTR replicas



NFS shared volume configuration for the DTR nodes are shown below. NFS back-end and configuration details have been omitted for simplicity.

Docker UCP Client Bundle

Client bundle is downloaded and check overall DEE cluster status.

```
# unzip ucp-bundle-admin.zip
Archive:  ucp-bundle-admin.zip
  extracting:  ca.pem
  extracting:  cert.pem
  extracting:  key.pem
  extracting:  cert.pub
  extracting:  env.cmd
  extracting:  env.sh
  extracting:  env.ps1

# docker node ls

```

ID	HOSTNAME	STATUS	AVAILABILITY
MANAGER STATUS			
90v3go6w08rdciwkncl7e22sd *	DEE-Ctrl-1.cisco.com	Ready	Active
Leader			
e9jenb4er830ypaekfixqg4ew	DEE-Ctrl-2.cisco.com	Ready	Active
Reachable			
96ej631ez1y3du9yzwqoqsaj0	DEE-Ctrl-3.cisco.com	Ready	Active
Reachable			
pnfytlrh30bzguc39vwza4v4	DEE-DTR-1.cisco.com	Ready	Active
v3m2einlpfridrmmt77siln9d	DEE-DTR-2.cisco.com	Ready	Active
yom7k0oog2dby754zhe9l4o5x	DEE-DTR-3.cisco.com	Ready	Active
6deagroubnz0pqnsלבabtlnxv	DEE-Wrk-1.cisco.com	Ready	Active
klaun7oufkwur9199jr6whce	DEE-Wrk-2.cisco.com	Ready	Active
qgmwrg58bxjkl0w8zismht5hc	DEE-Wrk-3.cisco.com	Ready	Active
pydytsdik4inzj5grb3kj72	DEE-Wrk-4.cisco.com	Ready	Active

```

# docker info
Containers: 121
  Running: 74
  Paused: 0
  Stopped: 47
Images: 66
Server Version: ucp/2.2.4
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint, whitelist
Nodes: 10
<snip>
<snip>
Cluster Managers: 3
DEE-Ctrl-1.cisco.com: Healthy
  └─ Orca Controller: https://10.65.122.61:443
  └─ Classic Swarm Manager: tcp://10.65.122.61:2376
  └─ Engine Swarm Manager: tcp://10.65.122.61:12376
  └─ KV: etcd://10.65.122.61:12379
DEE-Ctrl-2.cisco.com: Healthy
  └─ Orca Controller: https://10.65.122.62:443
  └─ Classic Swarm Manager: tcp://10.65.122.62:2376
  └─ Engine Swarm Manager: tcp://10.65.122.62:12376
  └─ KV: etcd://10.65.122.62:12379
DEE-Ctrl-3.cisco.com: Healthy
  └─ Orca Controller: https://10.65.122.63:443
  └─ Classic Swarm Manager: tcp://10.65.122.63:2376
  └─ Engine Swarm Manager: tcp://10.65.122.63:12376
  └─ KV: etcd://10.65.122.63:12379

```

```
<snip>  
<snip>
```

Contiv Installation

There are two methods to install Contiv on a pre-installed Docker EE cluster nodes in native Swarm mode. One method is to download Contiv v2plugging image from the Docker store and complete the installation as per the installation steps. In this method of installation, etcd cluster and Open V-Switch (OVS) should be configured on the Docker EE cluster nodes prior to the v2plugin installation. The other recommended method is to use Contiv Installer sourced from Contiv GitHub. The Contiv Installer is inclusive of all the dependencies such as etcd cluster and OVS configurations. The installer pulls Contiv v2plugin image from the Docker store, which runs as Docker plugin on the Docker EE cluster nodes. The installer pulls the other Contiv binaries such as `netctl` from Contiv GitHub and installs those on the cluster nodes depending on the roles assigned to the nodes.

Contiv installer can be run either from a build/web-server or from one of the cluster nodes itself. While installing Contiv from one of the cluster nodes, make sure it can connect to all other nodes through password-less SSH access. Contiv installer comes with Ansible playbook and is executed via installer scripts. Installation program runs as a container and requires Docker Engine to be available on the host on which the installer runs.

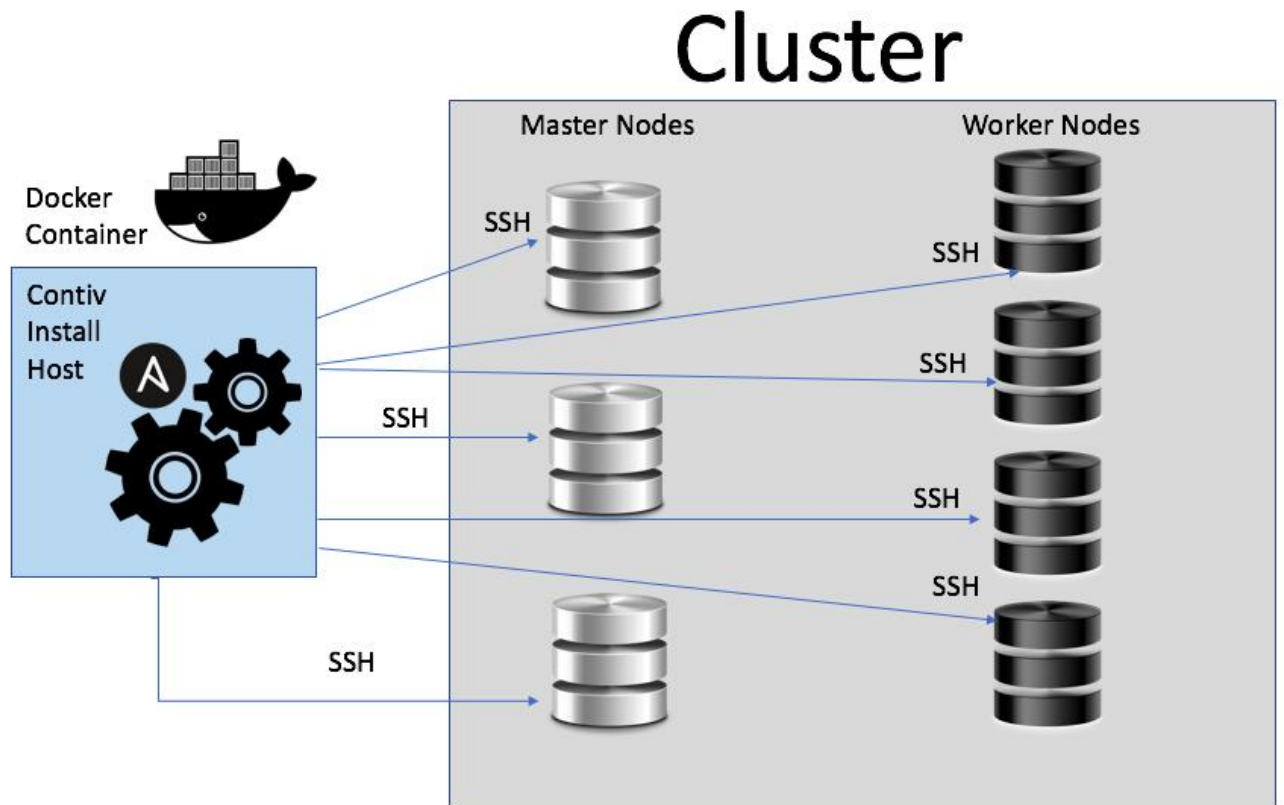


Docker has certified Contiv v2plugin 1.1.7 and is available for download at:

<https://store.docker.com/plugins/contiv>

Contiv installer release versions are maintained at: <https://github.com/contiv/install/releases>

Figure 29 Contiv Installation Process



For steps on installing Contiv in the recommended method using the Contiv installer, complete the following:

1. Generate SSH keys on one of the Swarm cluster node and populate it rest of the nodes:
5. [root@DEE-Ctrl1-1 ~]# ssh-keygen
2. Copy SSH key to rest of the cluster nodes -


```
# for i in 61 62 63 64 65 66 67 68 69 70;do echo 10.65.122.$i;ssh-copy-id
root@10.65.122.$i;done;
# for i in 61 62 63 64 65 66 67 68 69 70;do echo 10.65.122.$i;ssh root@10.65.122.$i
uptime;done;
10.65.122.61
 23:52:01 up  3:08,  2 users,  load average: 0.20, 0.21, 0.27
<snip>
```
3. Download the installer bundle. Download the full Contiv install and unzip the bundle -


```
# curl -L -O https://github.com/contiv/install/releases/download/1.1.7/contiv-full-
1.1.7.tgz
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100  611    0  611    0    0   410    0  --:--:--  0:00:01  --:--:--  411
100 113M 100 113M    0    0 151k    0  0:12:42  0:12:42  --:--:-- 96411

# tar -zxvf contiv-full-1.1.7.tgz
```
4. Navigate to contiv-1.1.7/install/ansible directory and edit 'cfg.yml' to define cluster host inventory and connection information. In this file, Contiv master and worker nodes are defined along with their

control and container data path network interface. As described in the earlier sections, second interface is used for container data path exclusively -

```
CONNECTION_INFO:
 10.65.122.61:
   role: master
   control: eno5
   data: eno6
 10.65.122.62:
   role: master
   control: eno5
   data: eno6
 10.65.122.63:
   role: master
   control: eno5
   data: eno6
 10.65.122.64:
   control: eno5
   data: eno6
 10.65.122.65:
   control: eno5
   data: eno6
 10.65.122.66:
   control: eno5
   data: eno6
 10.65.122.67:
   control: eno5
   data: eno6
 10.65.122.68:
   control: eno5
   data: eno6
 10.65.122.69:
   control: eno5
   data: eno6
 10.65.122.70:
   control: eno5
   data: eno6
```

5. On the same location the environmental variables are added to `env.json` file with proxy settings (if any).

Proxy settings are required in our environment -

```
"env":{ "http_proxy":"http://64.102.255.40:8080",
"HTTP_PROXY":"http://64.102.255.40:8080", "https_proxy":"http://64.102.255.40:8080",
"no_proxy":"127.0.0.1,localhost,netmaster,10.65.122.61,10.65.122.62,10.65.122.63,10.65.122.64,10.65.122.65,10.65.122.66,10.65.122.67,10.65.122.68,10.65.122.69,10.65.122.70,DEE-Ctrl-1,DEE-Ctrl-2,DEE-Ctrl-3,DEE-DTR-1,DEE-DTR-2,DEE-DTR-3,DEE-Wrk-1,DEE-Wrk-2,DEE-Wrk-3,DEE-Wrk-4"},
```

6. Install Contiv on the existing Swarm cluster nodes. Contiv v2Plugin is installed using host definition as captured in `cfg.yml` -

```
#!/install/ansible/install_swarm.sh -f install/ansible/cfg.yml -u root -e ~/.ssh/id_rsa -p
```

Installation is complete

```
=====
```

```
Please export DOCKER_HOST=tcp://10.65.122.61:2375 in your shell before proceeding
Contiv UI is available at https://10.65.122.61:10000
Please use the first run wizard or configure the setup as follows:
Configure forwarding mode (optional, default is bridge).
netctl global set --fwd-mode routing
Configure ACI mode (optional)
```

```
netctl global set --fabric-mode aci --vlan-range <start>-<end>
Create a default network
netctl net create -t default --subnet=<CIDR> default-net
For example, netctl net create -t default --subnet=20.1.1.0/24 default-net
```

```
=====
```

7. Validate Contiv plugin status -

```
# cat /var/log/contiv/plugin_bootup.log
2017-12-02T18:56:13Z|00001|vlog|INFO|opened log file /var/log/contiv/ovs-db.log
2017-12-02T18:56:13Z|00001|vlog|INFO|opened log file /var/log/contiv/ovs-vswitchd.log
Waiting for netmaster to be ready for connections
Netmaster ready for connections, setting forward mode to bridge
Forward mode is set
n-if=eno6 -cluster-store=etcd://localhost:2379 -ctrl-ip=10.65.122.61
/netmaster -plugin-name=contiv/v2plugin:1.1.7 -cluster-mode=swarm-mode -cluster-
store=etcd://localhost:2379 -control-url=10.65.122.61:9999

# netctl global info
Fabric mode: default
Forward mode: bridge
ARP mode: proxy
Vlan Range: 1-4094
Vxlan range: 1-10000
Private subnet: 172.19.0.0/16

# ansible DEE-Nodes -m shell -a "docker plugin ls"
DEE-Ctrl-1 | SUCCESS | rc=0 >>
ID                NAME                DESCRIPTION
ENABLED
7d63297e3b61     docker/telemetry:1.0.0.linux-x86_64-stable  Docker Inc. metrics
exporter         true
413db4eb861e     contiv/v2plugin:1.1.7  Contiv network plugin
for Docker      true

DEE-Ctrl-2 | SUCCESS | rc=0 >>
ID                NAME                DESCRIPTION
ENABLED
850835e60af1     docker/telemetry:1.0.0.linux-x86_64-stable  Docker Inc. metrics
exporter         true
2ac2ae595179     contiv/v2plugin:1.1.7  Contiv network plugin
for Docker      true

DEE-Ctrl-3 | SUCCESS | rc=0 >>
ID                NAME                DESCRIPTION
ENABLED
1a551fe25efa     contiv/v2plugin:1.1.7  Contiv network plugin
for Docker      true
```

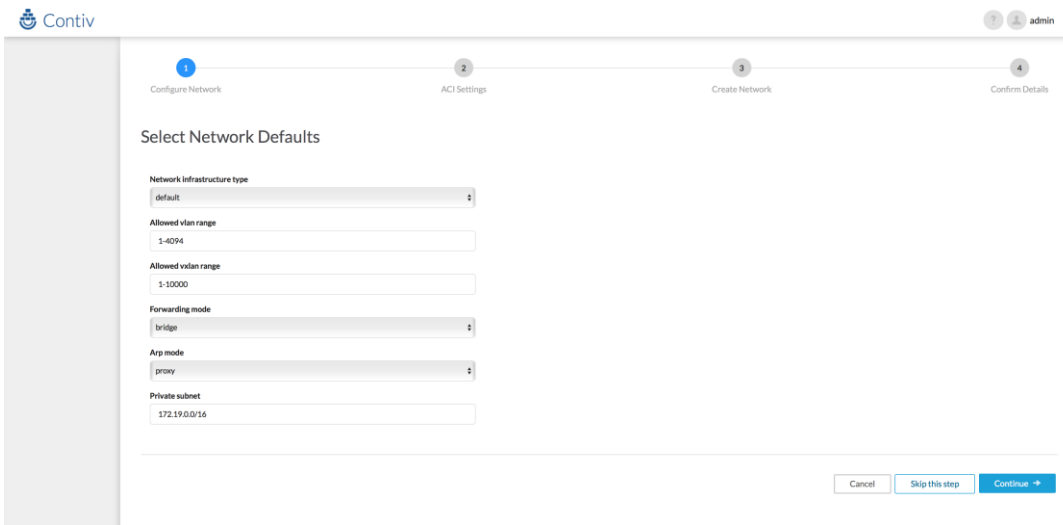
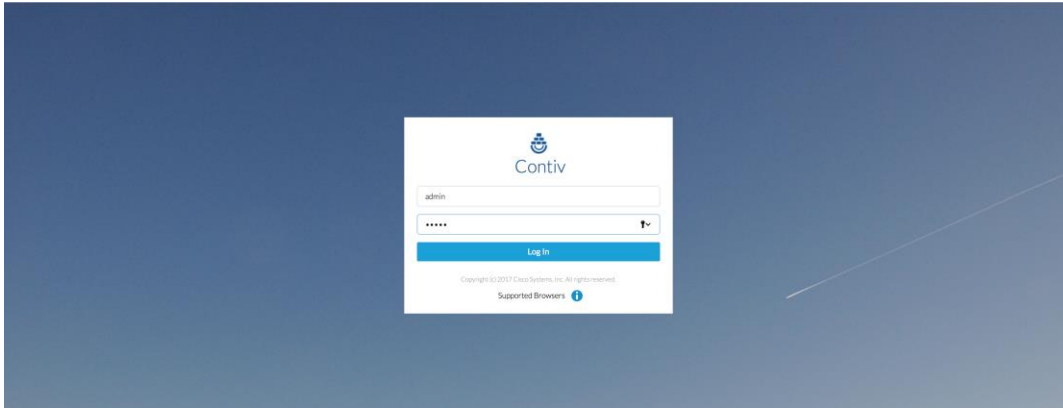
8. Validate etcd cluster state -

```
# ansible DEE-Nodes -m shell -a "etcdctl cluster-health"
DEE-Ctrl-1 | SUCCESS | rc=0 >>
member 7ebc124ca61e9cf1 is healthy: got healthy result from http://10.65.122.62:2379
member 948a257d00473f21 is healthy: got healthy result from http://10.65.122.63:2379
member e836c74d6a6071c4 is healthy: got healthy result from http://10.65.122.61:2379
cluster is healthy

# ansible DEE-Nodes -m shell -a "etcdctl member list"
DEE-Ctrl-3 | SUCCESS | rc=0 >>
7ebc124ca61e9cf1: name=node3 peerURLs=http://10.65.122.62:2380 client-
URLs=http://10.65.122.62:2379,http://10.65.122.62:4001 isLeader=false
948a257d00473f21: name=node2 peerURLs=http://10.65.122.63:2380 client-
URLs=http://10.65.122.63:2379,http://10.65.122.63:4001 isLeader=false
```

```
e836c74d6a6071c4: name=node1 peerURLs=http://10.65.122.61:2380 clientURLs=http://10.65.122.61:2379,http://10.65.122.61:4001 isLeader=true
```

9. Validate Contiv UI/auth_proxy Access -



At times there could be issues while installing Contiv v2plugin. Follow this GitHub entry for latest fixes:
<https://github.com/contiv/install/issues/340>

Validation

To validate this solution, tests were conducted on functional, HA and Scale aspects. Feature functional tests include routine container life-cycle management operations (create/delete and start/stop containers) through Docker UCP client bundle and UI.

Contiv provides interfaces to interact with its services through `netctl` CLI and `auth_proxy` UI. This solution document is focused on using CLI interface. Contiv UI was used only to validate that it covers the workflow without any issues. Contiv CLI can be executed from any of the Contiv masters, as masters share complete state information among the cluster members seamlessly. Contiv UI can also be accessed via any of the master IPs as URL - `https://<any of the master node IP>:10000`. Contiv UI needs to be configured with external load balancer, same as Docker UCP and DTR. Example configuration of external load-balancer - haproxy is shown in the Appendix section.



You can find the Contiv documentation on concepts, user guide and administration guide at: <http://contiv.github.io/documents/>

Docker UCP client requires Docker Toolbox to be installed on 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 the following 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/>

Application Container Deployment Using Contiv

Contiv Network Back-end without Contiv Policy Rules

In this section, Contiv network is created without the network policy applied to the created network. The deployed containers consuming this network were tested in a multi-host environment for network connectivity.

Docker Enterprise Edition with native swarm mode uses third party driver developed with CNM model. This requires driver specific options to be set through `opt` tags. Setting the tag enables Docker to create a network, which is mapped with the third-party network through a plugin.

When a Contiv back-end network is used for creating Docker network, the Contiv policy framework will not be applicable on the Docker network. Use case for this scenario exists when users want to use only Contiv back-end network without a policy model.

All Contiv CLI commands works on any of the Contiv master nodes.

1. First setting up Contiv global settings -

```
# netctl global info
Fabric mode: default
Forward mode: bridge
ARP mode: proxy
Vlan Range: 1-4094
Vxlan range: 1-10000
Private subnet: 172.19.0.0/16
```

2. Allowed VLAN list on the Contiv vNIC is 1001-1005. VLAN range can be changed in Contiv global parameters as shown below -

```
# netctl global set --vlan-range 1001-1005
```

```
# netctl global info
Fabric mode: default
Forward mode: bridge
ARP mode: proxy
Vlan Range: 1001-1005
Vxlan range: 1-10000
Private subnet: 172.19.0.0/16
```

Fabric mode = default which is for standalone, non-ACI mode
Forwarding mode = bridge as against L3 routed mode

This solution uses default mode for both Fabric and Forwarding.

3. Create Contiv network as below with default tenant -

```
# netctl tenant list
```

```
Name
-----
default
```

```
# netctl network create --nw-type data --encap vlan --pkt-tag 1001 --subnet
100.100.100.0/24 --gateway 100.100.100.254 --nw-tag default.test contiv.test
Creating network default:contiv.test
```

```
# netctl network list
```

Tenant	Network	Nw Type	Encap type	Packet tag	Subnet	Gateway
IPv6Subnet	IPv6Gateway	Cfgd Tag				
-----	-----	-----	-----	-----	-----	-----
default	contiv.test	data	vlan	1001	100.100.100.0/24	
	100.100.100.254			default.test		

Important point to note -

--encap = vlan - This can be either vlan or vxlan. In our solution, L2 VLAN forwarding mode preferred mode

--nw-type = data - either data or infra. We will be using for all Contiv networks, type as data only

--pkt-tag = 1001 - vLAN ID

--nw-tag = default.test - For all Contiv network to be consumed by `docker service` in native swarm mode, network-tag is essential for mapping Contiv network back-end with Docker network

--gateway = SVI interface IP, as created at aggregation layer

contiv.test = is the network name

```
# netctl network create --nw-type data --encap vlan --pkt-tag 1002 --subnet
101.101.101.0/24 --gateway 101.101.101.254 --nw-tag default.test1 contiv.test1
Creating network default:contiv.test1
```

```
# netctl network ls
```

Tenant	Network	Nw Type	Encap type	Packet tag	Subnet	Gateway
IPv6Subnet	IPv6Gateway	Cfgd Tag				
-----	-----	-----	-----	-----	-----	-----

```

default   contiv.test   data   vlan   1001   100.100.100.0/24
100.100.100.254
default   contiv.test1  data   vlan   1002   101.101.101.0/24
101.101.101.254
default.test1

```

4. Create Docker network to use Contiv network at the back-end with network-tag as a mapping parameter -

```

# docker network create -d contiv/v2plugin:1.1.7 -o contiv-tag=default.test --ipam-opt
contiv-tag=default.test --ipam-driver contiv/v2plugin:1.1.7 contiv-test

17dgqztf8y8qwqvz6616g4c1

# docker network create -d contiv/v2plugin:1.1.7 -o contiv-tag=default.test1 --ipam-opt
contiv-tag=default.test1 --ipam-driver contiv/v2plugin:1.1.7 contiv-test1

0eh5rw54kj458hb146e8ik5kj

```

For docker network create command, following parameters are needed
-d = driver - which is Contiv in our case
-o = Option map for mapping into Contiv/driver network
--ipam-driver & opt = Which is again Contiv driver

5. Verify docker network so created has the correct driver -

```

# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
7cc18d93bdc4       bridge              bridge              local
17dgqztf8y8        contiv-test         contiv/v2plugin:1.1.7  swarm
0eh5rw54kj45       contiv-test1        contiv/v2plugin:1.1.7  swarm
1f3eff4e534f       docker_gwbridge     bridge              local
syep5e2tgy8v       dtr-ol              overlay             swarm
7f75047d8776       host                host                local
iazypf0vtpg        ingress             overlay             swarm
29369da55420       none                null                local

```

'contiv-test/test1' network is showing the correct driver and SCOPE is set to swarm. This means that the network can be used across all the cluster nodes.

6. Inspect the created Docker network and verify if the correct IPAM driver is shown-

```

# docker network inspect contiv-test
[
  {
    "Name": "contiv-test",
    "Id": "17dgqztf8y8qwqvz6616g4c1",
    "Created": "0001-01-01T00:00:00Z",
    "Scope": "swarm",
    "Driver": "contiv/v2plugin:1.1.7",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "contiv/v2plugin:1.1.7",
      "Options": {
        "com.docker.network.ipam.serial": "true",
        "contiv-tag": "default.test"
      }
    },
    "Config": []
  },
  "Internal": false,
  "Attachable": false,
  "Ingress": false,
  "ConfigFrom": {
    "Network": ""
  },
  "ConfigOnly": false,
  "Containers": null,

```

```

        "Options": {
            "contiv-tag": "default.test"
        },
        "Labels": null
    }
]

# netctl network inspect contiv.test
{
    "Config": {
        "key": "default:contiv.test",
        "cfgdTag": "default.test",
        "encap": "vlan",
        "gateway": "100.100.100.254",
        "networkName": "contiv.test",
        "nwType": "data",
        "pktTag": 1001,
        "subnet": "100.100.100.0/24",
        "tenantName": "default",
        "link-sets": {},
        "links": {
            "Tenant": {
                "type": "tenant",
                "key": "default"
            }
        }
    },
    "Oper": {
        "allocatedIPAddresses": "100.100.100.254",
        "availableIPAddresses": "100.100.100.1-100.100.100.253",
        "networkTag": "default.test",
        "pktTag": 1001
    }
}

```

7. Now create an SVI on the aggregation layer. In the topology, Nexus 9000 switch is configured and has the SVIs created on them -

```

Docker-B# config t
Enter configuration commands, one per line. End with CNTL/Z.
Docker-B(config)# interface vlan 1001

```

```

Docker-B# config t
Enter configuration commands, one per line. End with CNTL/Z.
Docker-B(config)# interface vlan 1002
Docker-B(config-if)# ip address 101.101.101.254 255.255.255.0
Docker-B(config-if)# no shutdown
Docker-B(config-if)# end
Docker-B#

```

8. Now provision services for both of these networks to verify connectivity -

```

$ docker service create --name alpine-1 --network contiv-test alpine sleep 10000
tgq7wukhan05bcw599ca51fyv

```

```

$ docker service create --name alpine-2 --network contiv-test1 alpine sleep 10000
udjrlm2us2aovgifsxkgu2ou2

```

So, two services were created with both the Contiv network back-ends: contiv-test and contiv-test1.

9. Now scale the services so that containers are running on multiple nodes -

```

$ docker service ls

```

ID	NAME	MODE	REPLICAS	IMAGE
tgq7wukhan05bcw599ca51fyv	alpine-1	replicated	1/1	alpine:latest

```

udjrlm2us2ao      alpine-2      replicated    1/1          al-
pine:latest
qonpjlo5gtpy     ucp-agent    global       10/10       dock-
er/ucp-agent:2.2.4
myha6e10r41u     ucp-agent-s390x  global      0/0         dock-
er/ucp-agent-s390x:2.2.4
jt6nuoq68hla     ucp-agent-win  global       0/0         dock-
er/ucp-agent-win:2.2.4

```

```

$ docker service scale alpine-1=2
alpine-1 scaled to 2

```

```

$ docker service scale alpine-2=2
alpine-2 scaled to 2

```

```

$ docker service ls
ID                NAME                MODE                REPLICAS        IMAGE
PORTS
tgq7wukhan05     alpine-1            replicated          2/2             al-
pine:latest
udjrlm2us2ao     alpine-2            replicated          2/2             al-
pine:latest
qonpjlo5gtpy     ucp-agent           global              10/10           dock-
er/ucp-agent:2.2.4
myha6e10r41u     ucp-agent-s390x    global              0/0             dock-
er/ucp-agent-s390x:2.2.4
jt6nuoq68hla     ucp-agent-win      global              0/0             dock-
er/ucp-agent-win:2.2.4

```

10. Find out where the individual containers are running -

```

$ docker service ps alpine-1
ID                NAME                IMAGE                NODE                DE-
SIRED STATE      CURRENT STATE      ERROR                PORTS              Run-
khhh2j5ehtxt     alpine-1.1         alpine:latest       DEE-Wrk-3.cisco.com Running 7 minutes ago
v9eut1z7z4p5     alpine-1.2         alpine:latest       DEE-Wrk-4.cisco.com Running about a minute ago
ning

```

```

$ docker service ps alpine-2
ID                NAME                IMAGE                NODE                DE-
SIRED STATE      CURRENT STATE      ERROR                PORTS              Run-
1uil2mfkjply     alpine-2.1         alpine:latest       DEE-Wrk-2.cisco.com Running 6 minutes ago
q1j7huqfrbsh     alpine-2.2         alpine:latest       DEE-Wrk-1.cisco.com Running about a minute ago
ning

```

From the above example it is clear that the containers are distributed across all 4 worker nodes.

11. To cross verify the IP addresses these containers have obtained, login to the containers from both the alpine 1 and alpine 2 services on different Contiv networks as shown below -

```

$ docker ps
CONTAINER ID      IMAGE                COMMAND              CREATED
STATUS           PORTS              NAMES              DEE-
7f08a6dd48de     alpine:latest       "sleep 10000"       7 minutes
ago             Up 7 minutes      DEE-
Wrk-1.cisco.com/alpine-2.2.q1j7huqfrbshaymftk6xgyrcl
a8bf50b6ece6     alpine:latest       "sleep 10000"       7 minutes
ago             Up 7 minutes      DEE-
Wrk-4.cisco.com/alpine-1.2.v9eut1z7z4p5dhlx1ck2qdt65
6b64685a7055     alpine:latest       "sleep 10000"       11
minutes ago    Up 11 minutes    DEE-
DEE-Wrk-2.cisco.com/alpine-2.1.1uil2mfkjplyws5351apjldz3

```



```
f1c2293f1f78      alpine:latest      "sleep 10000"      12
minutes ago      Up 12 minutes
DEE-Wrk-3.cisco.com/alpine-1.1.khhh2j5ehtxt53zg00q2kcm3y
```

```
$ docker exec -it 7f08a6dd48de /bin/sh
/ # ifconfig -a
eth0      Link encap:Ethernet  HWaddr 02:02:65:65:65:03
          inet addr:101.101.101.3  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:8 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:648 (648.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
/ #
```

```
$ docker exec -it a8bf50b6ece6 /bin/sh
/ # ifconfig -a
eth0      Link encap:Ethernet  HWaddr 02:02:64:64:64:04
          inet addr:100.100.100.4  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:8 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:648 (648.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
/ #
```

12. The IP addresses from respective subnets are obtained correctly. And are reachable on the SVI/Gateway as expected -

```
Container Alpine-1.2
/ # ping 100.100.100.254
PING 100.100.100.254 (100.100.100.254): 56 data bytes
64 bytes from 100.100.100.254: seq=1 ttl=255 time=0.416 ms
64 bytes from 100.100.100.254: seq=2 ttl=255 time=0.270 ms
64 bytes from 100.100.100.254: seq=3 ttl=255 time=0.257 ms
64 bytes from 100.100.100.254: seq=4 ttl=255 time=0.401 ms
^C
```

```
Container Alpine-2.2
/ # ping 101.101.101.254
PING 101.101.101.254 (101.101.101.254): 56 data bytes
64 bytes from 101.101.101.254: seq=1 ttl=255 time=0.466 ms
64 bytes from 101.101.101.254: seq=2 ttl=255 time=0.363 ms
64 bytes from 101.101.101.254: seq=3 ttl=255 time=0.423 ms
```

^C

13. Check connectivity between the containers on different subnets. Since they are natively visible on the fabric, packet forwarding will happen at the SVI device as shown below -

```

/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:02:65:65:65:03
          inet addr:101.101.101.3  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:27 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2244 (2.1 KiB)  TX bytes:2520 (2.4 KiB)

/ # ping 100.100.100.4
PING 100.100.100.4 (100.100.100.4): 56 data bytes
64 bytes from 100.100.100.4: seq=0 ttl=63 time=0.715 ms
64 bytes from 100.100.100.4: seq=1 ttl=63 time=0.155 ms
64 bytes from 100.100.100.4: seq=2 ttl=63 time=0.130 ms
64 bytes from 100.100.100.4: seq=3 ttl=63 time=0.095 ms
^C
--- 100.100.100.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.095/0.273/0.715 ms
/ #

/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:02:64:64:64:04
          inet addr:100.100.100.4  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1808 (1.7 KiB)  TX bytes:1162 (1.1 KiB)

/ # ping 101.101.101.3
PING 101.101.101.3 (101.101.101.3): 56 data bytes
64 bytes from 101.101.101.3: seq=0 ttl=63 time=0.718 ms
64 bytes from 101.101.101.3: seq=1 ttl=63 time=0.166 ms
64 bytes from 101.101.101.3: seq=2 ttl=63 time=0.148 ms
^C
--- 101.101.101.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.148/0.344/0.718 ms
/ #

```

Both the containers residing on the two different subnets are able to reach.

14. Validate if Containers using Contiv networks are visible on the fabric -

```

Container alpine-2.2 Mac-address
/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:02:65:65:65:03

Container alpine-1.2 Mac-address
/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:02:64:64:64:04

Docker-B# sho ip arp

Flags: * - Adjacencies learnt on non-active FHRP router
       + - Adjacencies synced via CFSOE
       # - Adjacencies Throttled for Glean
       CP - Added via L2RIB, Control plane Adjacencies
       D - Static Adjacencies at-
          tached to down interface

```

```
IP ARP Table for context default
Total number of entries: 2
Address      Age      MAC Address      Interface
100.100.100.4 00:02:32 0202.6464.6404  Vlan1001
101.101.101.3 00:00:19 0202.6565.6503  Vlan1002
```

Mac-addresses are natively learned on aggregation layer, which is Nexus 9000 switch.
And additionally, these mac-addresses are learned on Fabric Interconnects as well -

```
Docker-B(nxos)# show mac address-table vlan 1001
Legend:
      * - primary entry, G - Gateway MAC, (R) - Routed MAC, O - Overlay MAC
      age - seconds since last seen,+ - primary entry using vPC Peer-Link
      VLAN      MAC Address      Type      age      Secure NTFY      Ports/SWID.SSID.LID
-----+-----+-----+-----+-----+-----+-----
* 1001      0202.6464.6402      dynamic  0          F      F      Veth1848
* 1001      0202.6464.6403      dynamic  0          F      F      Veth1848
* 1001      0202.6464.6404      dynamic  0          F      F      Veth1824
```

```
Docker-B(nxos)# show mac address-table vlan 1002
Legend:
      * - primary entry, G - Gateway MAC, (R) - Routed MAC, O - Overlay MAC
      age - seconds since last seen,+ - primary entry using vPC Peer-Link
      VLAN      MAC Address      Type      age      Secure NTFY      Ports/SWID.SSID.LID
-----+-----+-----+-----+-----+-----+-----
* 1002      0202.6565.6502      dynamic  0          F      F      Veth1836
* 1002      0202.6565.6503      dynamic  0          F      F      Veth1844
```

Contiv Network Back-end with Contiv Policy Rules

Docker Enterprise Edition with native swarm mode requires third-party driver support through `opt` map for driver specific options. This enables Contiv to apply policy model to the network it provides to the application containers. For this Contiv allows you apply `nw-tag` to the Contiv groups associated with Contiv network. Subsequently any policy associated with group allows us to apply network forwarding rule sets for the application containers. Contiv back-end network for Docker EE with policy model is the use case here.

For more information on Contiv Policy Model, refer:

<http://contiv.github.io/documents/networking/policies.html>

Following is the validation with results on Contiv policy model workflow:

- Created Contiv network without tag -


```
# netctl network create --nw-type data --encap vlan --pkt-tag 1001 --subnet
100.100.100.0/24 --gateway 100.100.100.254 contiv-test

Creating network default:contiv-test
```
- Created network policy and applied rules to it. In this example a policy was created to allow inbound access to tcp/80 and tcp/443 and deny all other traffic -


```
# netctl policy create web-policy
Creating policy default:web-policy

# netctl policy rule-add web-policy 1 -direction=in -protocol=tcp -action=deny

# netctl policy rule-add web-policy 2 -direction=in -protocol=tcp -port=80 -
action=allow -priority=10

# netctl policy rule-add web-policy 3 -direction=in -protocol=tcp -port=443 -
action=allow -priority=10
```

3. After defining a policy, it was associated with a group on an existing Contiv network -

```
# netctl group create -p web-policy -tag test-tag contiv-test web-group
Creating EndpointGroup default:web-group
```

4. Verified all Contiv objects are configured as expected -

```
# netctl network ls
Tenant      Network      Nw Type  Encap type  Packet tag  Subnet      Gateway
IPv6Subnet  IPv6Gateway  Cfgd Tag
-----
default     contiv-test  data     vlan        1001        100.100.100.0/24
100.100.100.254

# netctl group ls
Tenant      Group      Network      IP Pool  CfgdTag      Policies      Network profile
-----
default     web-group  contiv-test          test-tag      web-policy

# netctl policy ls
Tenant      Policy
-----
default     web-policy

# netctl policy rule-ls web-policy
Incoming Rules:
Rule  Priority  From EndpointGroup  From Network  From IpAddress  To IpAddress  Proto-
col  Port  Action
----  -
1     1
0     deny
2     10
80    allow
3     10
443   allow
Outgoing Rules:
Rule  Priority  To EndpointGroup  To Network  To IpAddress  Protocol  Port  Action
-----
```

5. Created Docker network with contiv-tag -

```
# docker network create contiv-test -o contiv-tag=test-tag -d contiv/v2plugin:1.1.7 --
ipam-opt contiv-tag=test-tag --ipam-driver contiv/v2plugin:1.1.7
xawzx51r9teryajb3951rtvtj
```

6. See the list of Docker network to check if the Docker network is created with Contiv driver -

```
# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
7cc18d93bdc4       bridge             bridge              local
xawzx51r9ter        contiv-test        contiv/v2plugin:1.1.7  swarm
1f3eff4e534f        docker_gwbridge    bridge              local
syep5e2tgy8v        dtr-ol             overlay             swarm
7f75047d8776        host               host                local
iazypf0vtpg         ingress            overlay             swarm
29369da55420        none               null                local
```

7. Deployed `docker service` with Contiv network and scaled it to spread on multi-host -

```
$ docker service create --name contiv-alpine --network contiv-test alpine sleep 10000
i7y7qdh8se2mw9y2ukco08vgr
```

```
$ docker service scale contiv-alpine=4
contiv-alpine scaled to 4
```

```
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE
PORTS				
i7y7qdh8se2m	contiv-alpine	replicated	4/4	al-
pine:latest				
qonpj1o5gtpy	ucp-agent	global	11/11	dock-
er/ucp-agent:2.2.4				
myha6e10r41u	ucp-agent-s390x	global	0/0	dock-
er/ucp-agent-s390x:2.2.4				
jt6nuoq68hla	ucp-agent-win	global	0/0	dock-
er/ucp-agent-win:2.2.4				

```
$ docker service ps contiv-alpine
```

ID	NAME	IMAGE	NODE	DE-
SIRE0 STATE	CURRENT STATE	ERROR	PORTS	
dmt00ghi2c5j	contiv-alpine.1	alpine:latest	DEE-Wrk-5.cisco.com	Run-
ning	Running about a minute ago			
ze9rbtypems2	contiv-alpine.2	alpine:latest	DEE-Ctrl-1.cisco.com	Run-
ning	Running 40 seconds ago			
pgf1r1fky9aqz	contiv-alpine.3	alpine:latest	DEE-Wrk-1.cisco.com	Run-
ning	Running about a minute ago			
a5b8hka3evsd	contiv-alpine.4	alpine:latest	DEE-Wrk-2.cisco.com	Run-
ning	Running about a minute ago			

8. Logged in to containers deployed and validated Contiv policy rules for network resource access - Container contiv-alpine.4

```
$ docker exec -it 208258a9265d /bin/sh
```

```
# ifconfig -a
```

```
eth0      Link encap:Ethernet  HWaddr 02:02:64:64:64:05
          inet addr:100.100.100.5  Bcast:0.0.0.0  Mask:255.255.255.0
```

```
# nc -vl -p 80
```

```
listening on [::]:80 ...
```

```
connect to [::ffff:100.100.100.5]:80 from contiv-
```

```
alpine.3.pgfr1fky9aqzmc3j9872gzw8.contiv-test:45899 ([::ffff:100.100.100.4]:45899)
```

```
Container contiv-alpine.3
```

```
# nc -nzvw 1 100.100.100.5 80
```

```
100.100.100.5 (100.100.100.5:80) open
```

```
# nc -nzvw 1 100.100.100.5 81
```

```
nc: 100.100.100.5 (100.100.100.5:81): Operation timed out
```

This validates that the Containers cannot connect to each other, when they try to connect on the ports other than the allowed ports.

9. Container contiv-alpine.4, ran `nc` on port 81

```
# nc -vl -p 81
```

```
listening on [::]:81 ...
```

10. Container contiv-alpine.3, tried to connect to `nc` on port 81 on contiv-alpine.4

```
/ # nc -nzvw 1 100.100.100.5 81
```

```
nc: 100.100.100.5 (100.100.100.5:81): Operation timed out
```

Test Plan

Following feature functional, high-availability and scale tests were executed as part of the solution validation.

Functional Test Scenarios – Docker EE/ Contiv

1. Create network, subnet, VLAN, application profile and policy constructs
2. Create and deploy containers with Contiv network
3. Test the connectivity with all possible scenarios for L2 VLAN mode of operation under Contiv work-flow
4. Create Container-Application-Groups for Contiv network, Define Policies, Associate Policies to container-groups, ensure that the policies were enforced
5. Validate discovery of new end-points (new containers etc) at the fabric
6. While configuring various policies, networks, tenants, etc. validate the error messages and appropriate warning messages on CLI/GUI
7. Isolation policy and Bandwidth policy tests and validation
8. Service discovery tests – DNS/IPAM driver basic validation



Currently the containers deployed through 'docker service' do not get the DNS names. A defect has been raised with both Docker/Contiv.

For the issue on Contiv, see: <https://github.com/contiv/netplugin/issues/968>

For the issue on Docker, see: <https://github.com/docker/libnetwork/pull/1855>

9. Container data path and connectivity tests between different subnet via SVI/inter-vLAN routing at TOR switch
10. Multi-tier application container stack deployment with Contiv policy on different subnet and validate connectivity between the application stack components
11. Contiv workflow through Contiv UI with Docker EE Swarm mode validation



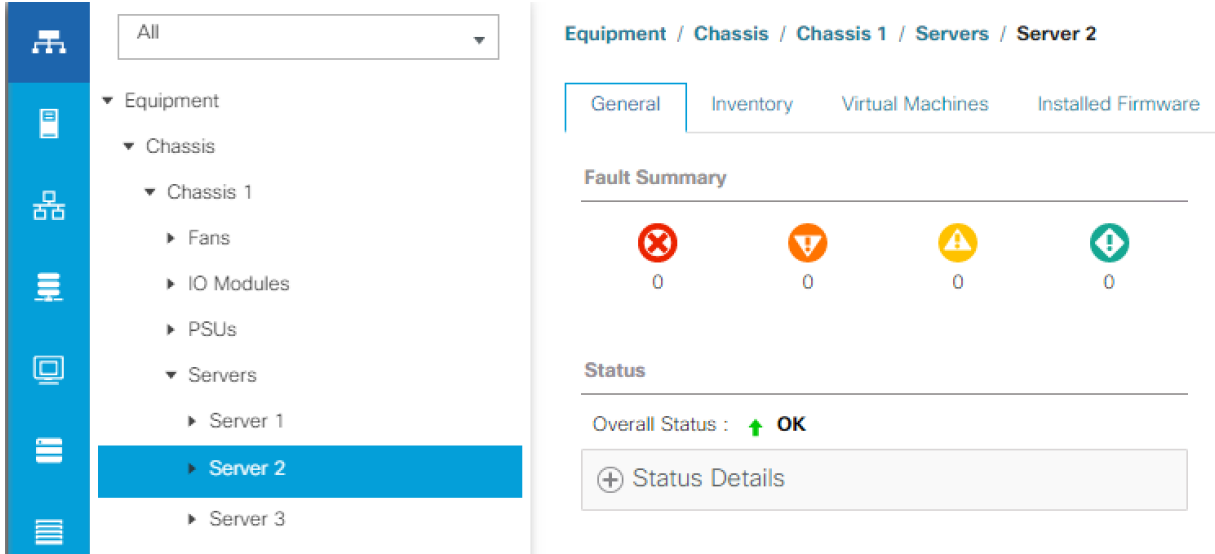
Contiv network deployment workflow currently works with Contiv CLI and Docker UCP client bundle. Contiv UI can also be used for creating Contiv backend network. However, the support for creating Contiv backend network on Docker UCP GUI is a feature currently under development; ticket number #9068 is prioritized to support this feature. Therefore, the workflows are currently validated using Docker UCP CLI in this CVD.

Scale Tests

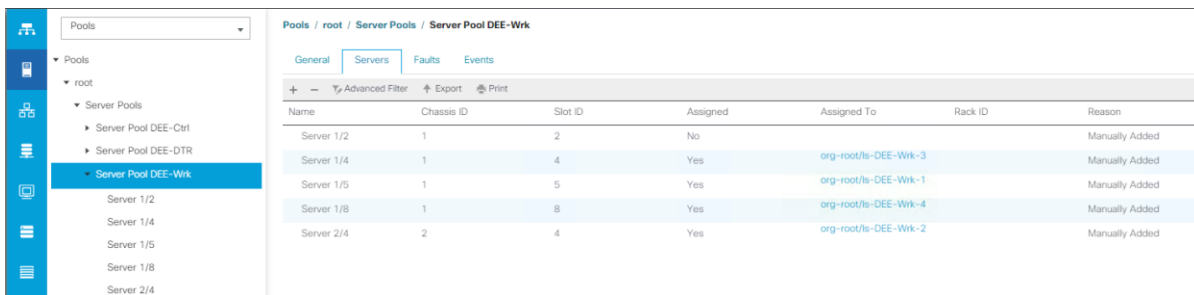
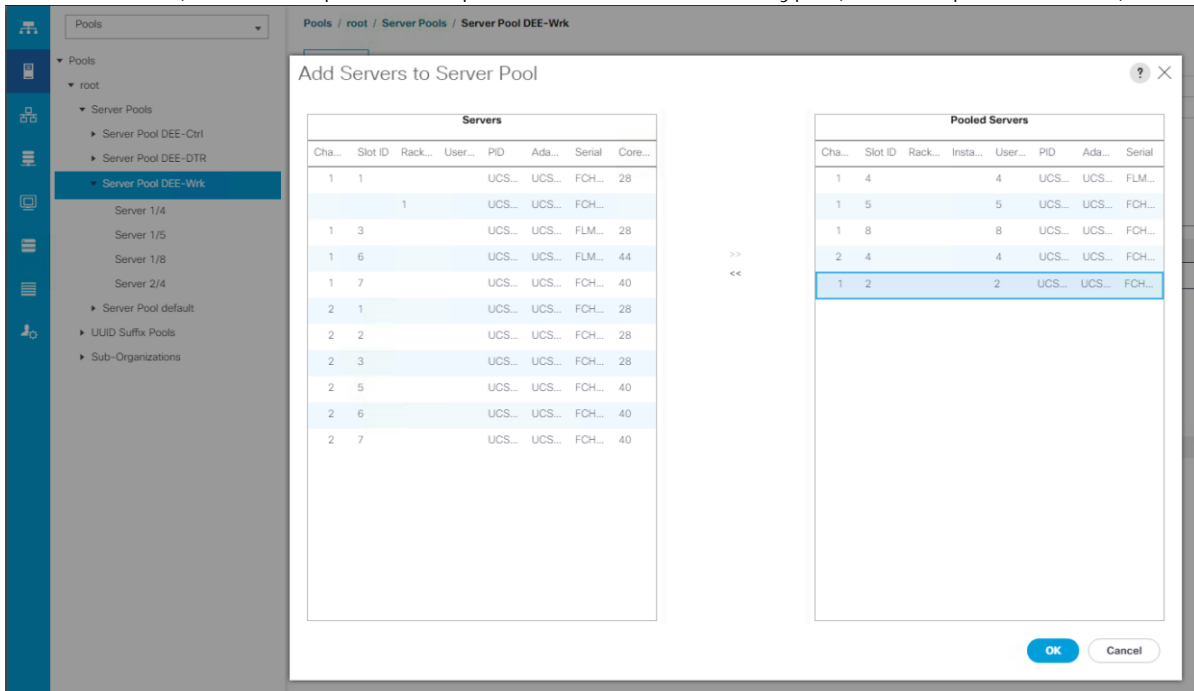
Cisco UCS infrastructure has a built-in mechanism to scale the deploy application environment through service profile templates and server pool concepts. It is very easy to scale-up the infrastructure. By inserting new blade hardware and adding it into the server pool, gets discovered automatically and 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 Enterprise pod. This procedure remains the same for any type of nodes, be it a UCP master, UCP worker or DTR node till the point where OS and Docker Engine is installed. 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 master/controllers, UCP/DTR worker nodes. Service profiles are instantiated from one of these service profile templates based on the need for scale of a particular type of node.

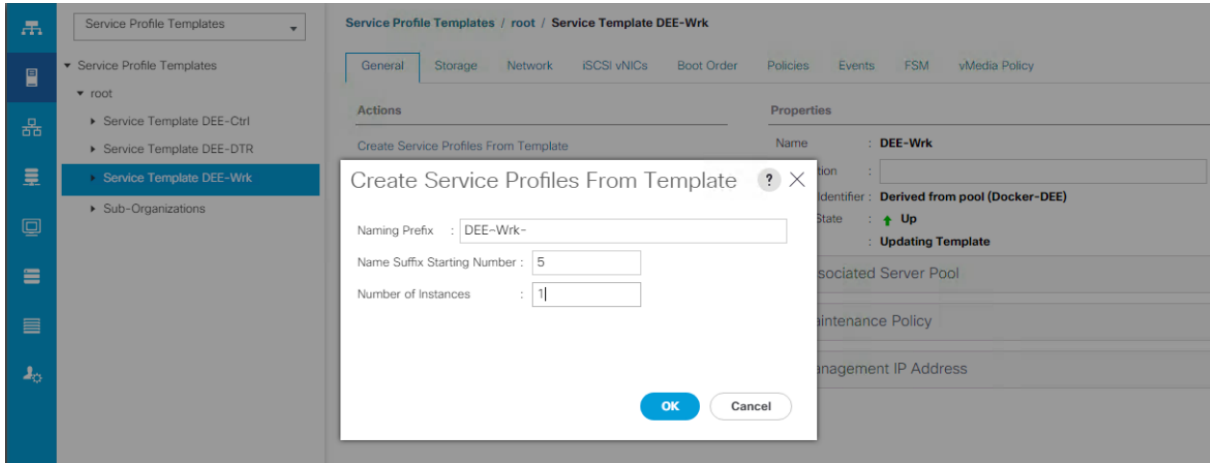
Ansible playbook completes post OS installation tasks including adding the node to the existing cluster based on node role. After Docker EE is up and running on the new node, Contiv installation is done, which adds the new node to the Contiv cluster. Currently, Contiv supports Contiv worker node addition seamlessly.

1. Insert a new blade in any available chassis slot and get it discovered:

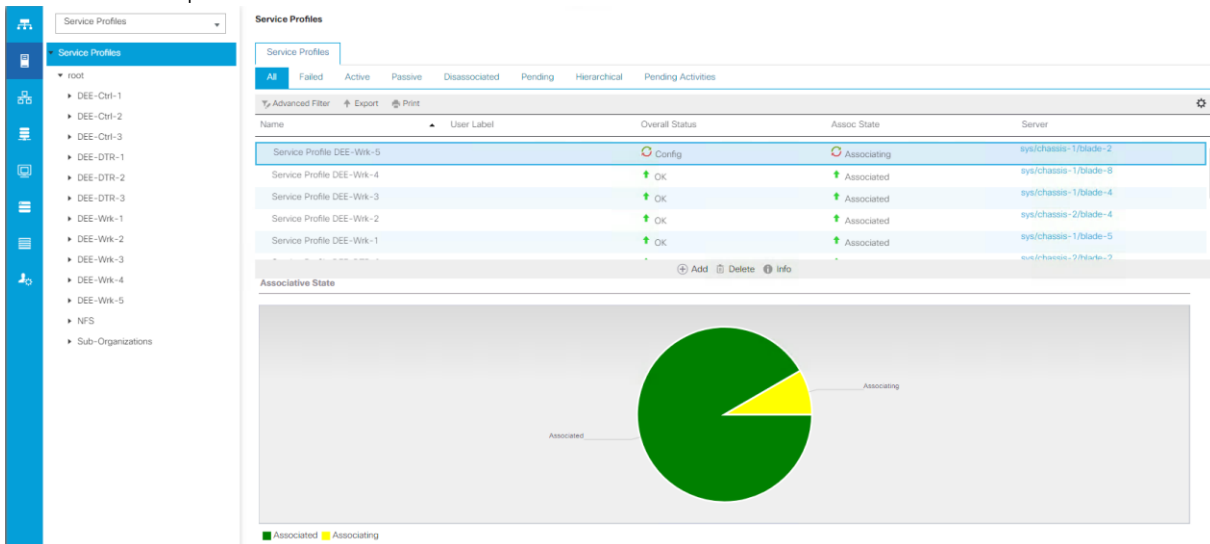


2. After successfully adding the node to the server pool, create new service profile (for example, DEE-Wrk-5) from the specific template based on the node type (for example, DEE-Wrk):

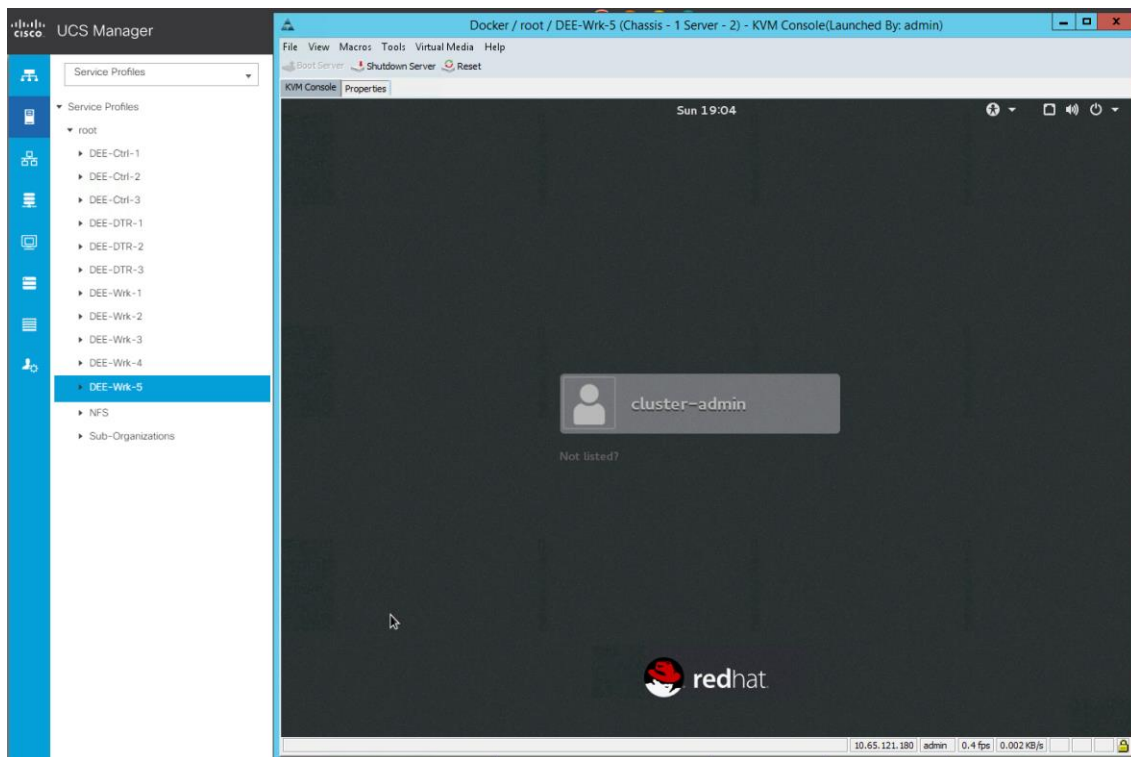
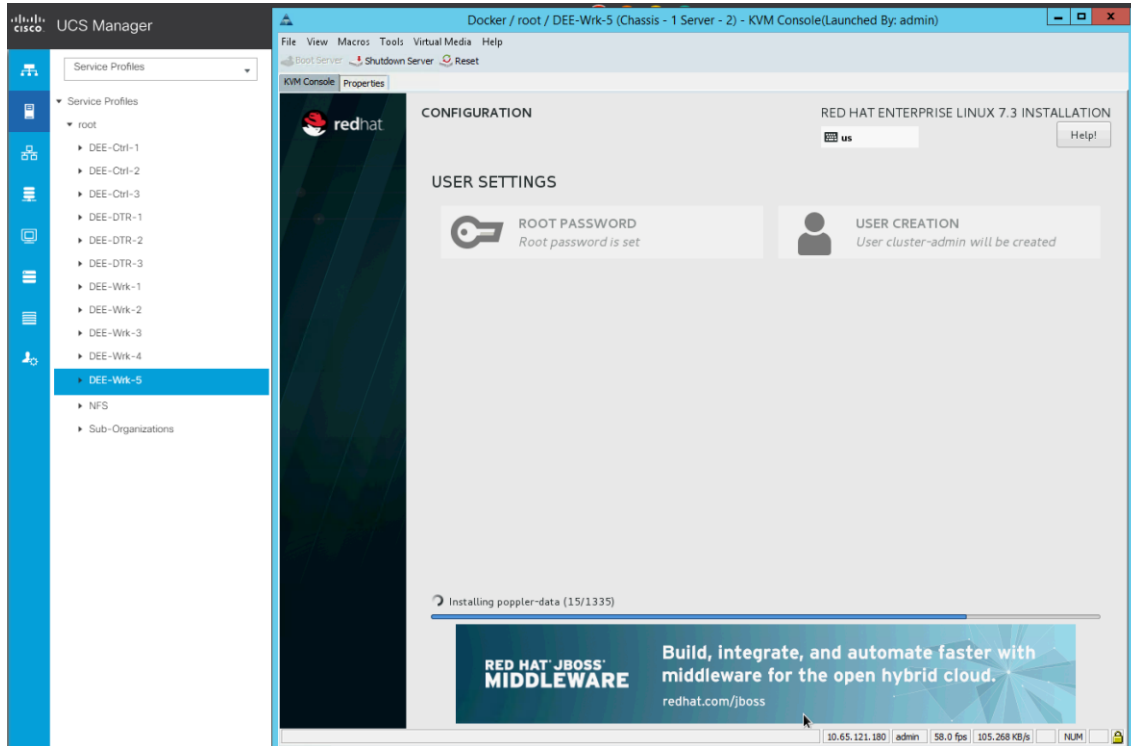




3. Service profile association will automatically kick-in as the template was already associated to the server pool:



4. While service profile is getting associated, kickstart HDD image was created for the new node. Steps outlined for the same in earlier sections.
5. After completion of successful service profile association our automated OS installation workflow will commence and install OS through vMedia mounted boot and kickstart images:



6. For post OS install tasks, modify the Ansible playbook to include roles/tasks for DEE-Worker node role to get the node attached to the existing Docker EE Swarm cluster:

```
PLAY [UCP-Mgr] *****
TASK [UCFwarm : capture swarm join token - worker] *****
changed: [DEE-Ctrl-1] => {"changed": true, "cmd": "docker swarm join-token worker | grep token | awk '{print $5}'", "delta": "0:00:00.057667", "end": "2017-12-03 20:22:26.787723", "rc": 0, "start": "2017-12-03 20:22:26.730056", "stderr": "", "stderr_lines": [], "stdout": "SWMTRN-1-4dqrjmkob3gwr75k510oyandzd8nqds2cm4veewyux4syuuxny-2gcpblnxk1cqm8fqhno6zbc8g", "stdout_lines": ["SWMTRN-1-4dqrjmkob3gwr75k510oyandzd8nqds2cm4veewyux4syuuxny-2gcpblnxk1cqm8fqhno6zbc8g"]}
PLAY [UCF-Wrk] *****
TASK [UCFworker : Node joining to the cluster as worker] *****
changed: [DEE-Wrk-5] => {"changed": true, "cmd": "docker swarm join --token SWMTRN-1-4dqrjmkob3gwr75k510oyandzd8nqds2cm4veewyux4syuuxny-2gcpblnxk1cqm8fqhno6zbc8g 10.65.122.61:2377", "delta": "0:00:00.322964", "end": "2017-12-03 20:22:27.338226", "rc": 0, "start": "2017-12-03 20:22:27.015262", "stderr": "", "stderr_lines": [], "stdout": "This node joined a swarm as a worker.", "stdout_lines": ["This node joined a swarm as a worker."]}
PLAY RECAP *****
DEE-Ctrl-1 : ok=1 changed=1 unreachable=0 failed=0
DEE-Wrk-5 : ok=64 changed=58 unreachable=0 failed=0
```

Status	Name	Role	Address	Engine	OS/Arch	CPU	Memory	Disk	Details
●	DEE-Wrk-5.cisco.com	worker	10.65.122.71	17.06.2-ee-6	linux/x86_64	-	-	-	Healthy UCP worker

- Once node joins the existing DEE Swarm cluster, the node is added to the Contiv inventory file 'cfg.yml' and run the installer to get Contiv installed on it.

```
Installation is complete
=====
Please export DOCKER_HOST=tcp://10.65.122.61:2375 in your shell before proceeding
Contiv UI is available at https://10.65.122.61:10000
Please use the first run wizard or configure the setup as follows:
Configure forwarding mode (optional, default is bridge).
netctl global set --fwd-mode routing
Configure ACI mode (optional)
netctl global set --fabric-mode aci --vlan-range <start>-<end>
Create a default network
netctl net create -t default --subnet=<CIDR> default-net
For example, netctl net create -t default --subnet=20.1.1.0/24 default-net
=====
```

```
[root@DEE-Ctrl-1 contiv-1.1.7]# ssh root@10.65.122.71 docker plugin ls
ID NAME DESCRIPTION ENABLED
d07bac078d70 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
d18dc71b9c75 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true
[root@DEE-Ctrl-1 contiv-1.1.7]# ssh root@10.65.122.71 etcdctl cluster-health
member 7ebc124ca61e9cf1 is healthy: got healthy result from http://10.65.122.62:2379
member 948a257d00473f21 is healthy: got healthy result from http://10.65.122.63:2379
member e836c74d6a6071c4 is healthy: got healthy result from http://10.65.122.61:2379
cluster is healthy
```

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 EE UCP master/controllers, UCP and DTR worker 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 Enterprise Edition

- UCP worker 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 Manager Master/Replica node reboot:
 - Tests passed. UCP management control plane through CLI/GUI was available for cluster/application container administration. UCP master node reboot did not impact container data-path and subsequent re-convergence of the cluster.
 - Same tests were performed with HAProxy external load-balancer in the mix. And UCP/DTR UI were accessible from the rest of the UCP master and DTR worker node, there were no service disruptions.
4. UCP Manager Master/Replica node shutdown:
 - Tests passed. Master/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 when the node was up.
 - Containers were deployed and the applications running on them were not disrupted on the UCP worker nodes during the unavailability of the UCP master node.
 - Same tests were performed with HAProxy external load-balancer in the mix. And UCP/DTR UI were accessible from the rest of the UCP master and DTR worker node, there were no service disruptions.
5. Docker Engine Service Restart on UCP worker 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.
8. Infrastructure component level service/process restart:
 - Docker EE services runs on cluster nodes as infrastructure containers except for the Docker EE Engine, which is a **'systemd' process controlled at** the operating system level through **'systemctl' control calls. Docker Engine restart** tests have been covered in step 7. As part of this test case, the key services and components of the software stack on each node type came up gracefully.

Tests Performed with Contiv

1. Contiv worker node reboot/shutdown:

While container applications were running using Contiv network and policies, Contiv worker node was rebooted. Swarm scheduler restarted those application containers on the surviving Contiv worker nodes. There was no delay or failure seen. Also `docker plugin ls` showed Contiv NetPlugin was coming back to enabled state after reboot/shutdown.
2. Contiv master node reboot/shutdown:

Same test as above performed on the Contiv master node. Contiv master nodes run `etcd` cluster as well for the state store KV database. In a 3 node master HA cluster only one master node failure can be sustained. With one master node reboot/shutdown scenarios were tested. All the services ran normally and no delay or failure seen in moving application containers and infrastructure services on the surviving master nodes. With one master node in failed state, new Contiv networks were successfully deployed and containers were able to consume them. Also with `etcd` cluster leader reboot/shutdown test, leadership/master role successfully migrated to one of the surviving master nodes. After the master node reboot/shutdown, when the node came up, the node was able to successfully join the cluster.

Tests Performed on Cisco UCS Infrastructure

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 inter-

nally 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.

Bill of Materials

The following infrastructure components are needed for UCS B-Series first architecture:

Table 12 BOM for first architecture

Component	Model	Quantity	Comments
Docker Enterprise Edition UCP Master/Controller Nodes	B200 M5 (UCSB-B200-M5-U)	3	<p>CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130)</p> <p>Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G1RS-H)</p> <p>Local Disks – 2 x 300 GB SAS disks for OS Boot & Docker Engine (UCS-HD300G10K12G)</p> <p>Network Card – 1x1340 VIC (UCSB-MLOM-40G-03)</p> <p>Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)</p>
Docker Enterprise Edition DTR Nodes	B200 M5 (UCSB-B200-M5-U)	3	<p>CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130)</p> <p>Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G1RS-H)</p> <p>Local Disks – 2 x 300 GB SAS disks for OS Boot & Docker Engine (UCS-HD300G10K12G)</p> <p>Network Card – 1x1340 VIC (UCSB-MLOM-40G-03)</p> <p>Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)</p>
Docker Enterprise Edition UCP Worker Nodes	B200 M5 (UCSB-B200-M5-U)	4	<p>CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130)</p> <p>Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G1RS-H)</p> <p>Local Disks – 2 x 300 GB SAS disks for OS Boot & Docker Engine (UCS-HD300G10K12G)</p> <p>Network Card – 1x1340 VIC (UCSB-MLOM-40G-03)</p>

			Raid Controller – Cisco MRAID 12 G SAS Controller (UCSB-MRAID12G)
Chassis	UCS 5108 (N20-C6508)	2	
IO Modules	IOM 2304 (UCS-IOM-2304)	4	
Fabric Interconnects	UCS 6332-16UP (UCS-FI-6332-16UP)	2	
Switches	Nexus 9396PX (N9K-C9396PX)	2	
Docker Server/On-Prem Subscription	Docker Enterprise Edition Subscription	1	<ul style="list-style-type: none"> • PIDs for Docker EE Standard for Linux Server with Business Critical Support/ Business Day Support: DSUB-EE-STANDARD-BC/ DSUB-EE-STANDARD-BD • PIDs for Docker EE Advanced for Linux Server with Business Critical Support/ Business Day Support: DSUB-EE-ADVANCED-BC/ DSUB-EE-ADVANCED-BD

The following infrastructure components are needed for UCS C-Series second architecture:

Table 13 BOM for second architecture

Component	Model	Quantity	Comments
Docker Enterprise Edition UCP Master/Controller + DTR Nodes	C220 M5 (UCSC-C220-M5SX)	3	<p>CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130)</p> <p>Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G2RS-H)</p> <p>Local Disks – 8 x 600 GB SAS disks for OS Boot and Docker Engine (UCS-HD600G15K12N)</p> <p>Network Card – 1x1385 VIC (UCSC-PCIE-C40Q-03)</p>

			Raid Controller – Cisco MRAID 12 G SAS Controller (UCSC-RAID-M5)
Docker Enterprise Edition UCP Worker Nodes	C220 M5 (UCSC-C220-M5SX)	1	<p>CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130)</p> <p>Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G2RS-H)</p> <p>Local Disks – 8 x 600 GB SAS disks for OS Boot and Docker Engine (UCS-HD600G15K12N)</p> <p>Network Card – 1x1385 VIC (UCSC-PCIE-C40Q-03)</p> <p>Raid Controller – Cisco MRAID 12 G SAS Controller (UCSC-RAID-M5)</p>
Fabric Interconnects	UCS 6332-16UP (UCS-FI-6332-16UP)	2	
Switches	Nexus 9396PX (N9K-C9396PX)	2	
Docker Server/On-Prem Subscription	Docker Enterprise Edition Subscription	1	<ul style="list-style-type: none"> • PIDs for Docker EE Standard for Linux Server with Business Critical Support/ Business Day Support: DSUB-EE-STANDARD-BC/ DSUB-EE-STANDARD-BD • PIDs for Docker EE Advanced for Linux Server with Business Critical Support/ Business Day Support: DSUB-EE-ADVANCED-BC/ DSUB-EE-ADVANCED-BD

Addendum

UCS with Contiv and DEE - Hybrid cluster with Baremetal and VMs

This solution has also been validated on Cisco UCS hardware platform running Docker/Contiv Manager and Worker nodes part of a mixed cluster having both bare metal and virtual machine hosts. This part of the **solution highlights Contiv's key feature**, which is to unify containers, VMs and bare metal hosts with a single networking fabric allowing container networks to be addressable from VMs and bare metal host networks. This enable us to move application workloads running inside containers through Contiv's application-oriented network policies across legacy bare metal and virtual hosting environments.

Mixed Cluster Solution Components

Solution validation on mixed cluster topology requires virtualization platform in addition to the base components listed **in the section "Solution Components"**. Solution validation on mixed cluster topology requires the following additional virtualization components:

- VMWare ESXi v6.5
- VMWare VCenter v6.5

Solution Design

For the mixed cluster topology, solution envisages VMWare ESXi hypervisor nodes to be added to the existing Docker UCP swarm mode cluster running on baremetal nodes. This solution uses two ESXi nodes with the same hardware and software components for provisioning virtual machines. Management of ESXi nodes, compute, network and storage needs for virtual machines are administered through VMWare vCenter.

Physical Topology

Following figures illustrate physical topologies for both Cisco UCS Blade and Rack server architectures.

Figure 30 Mixed cluster with B-series servers - First Architecture

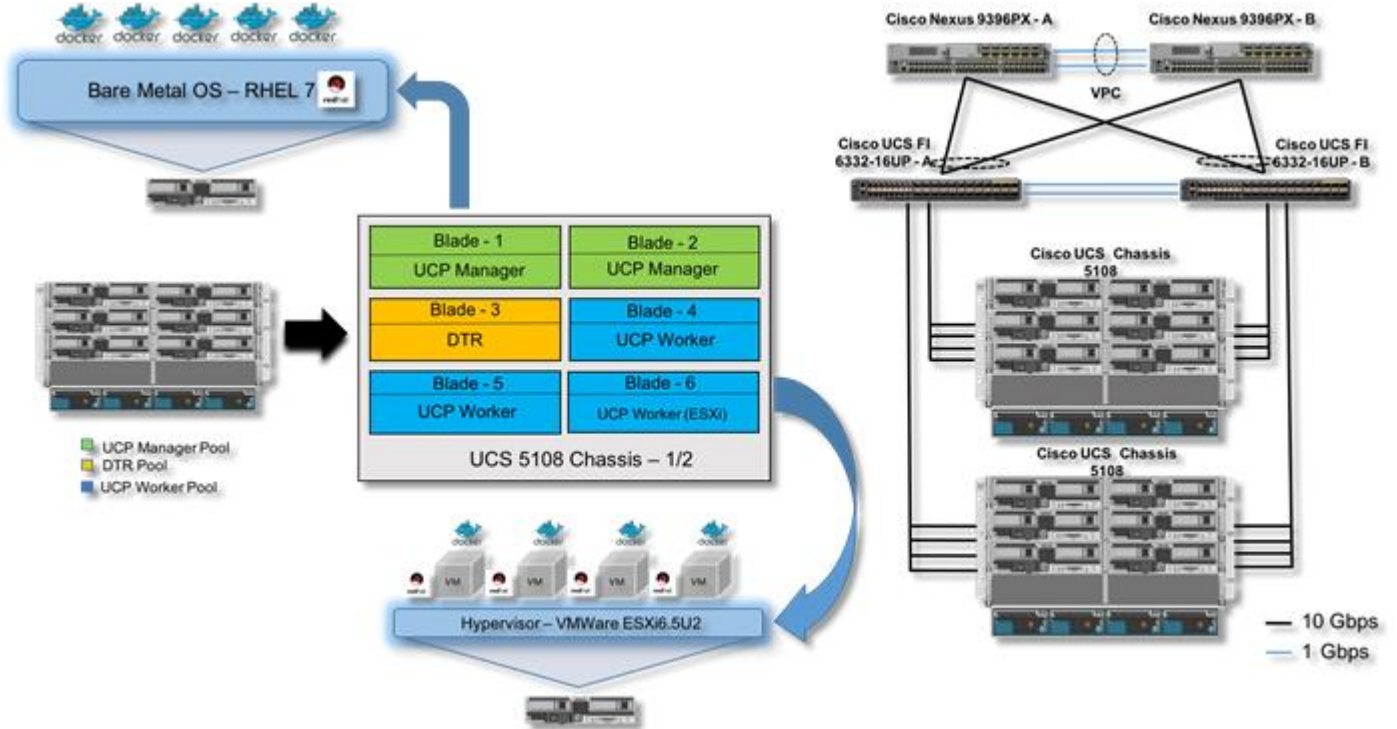
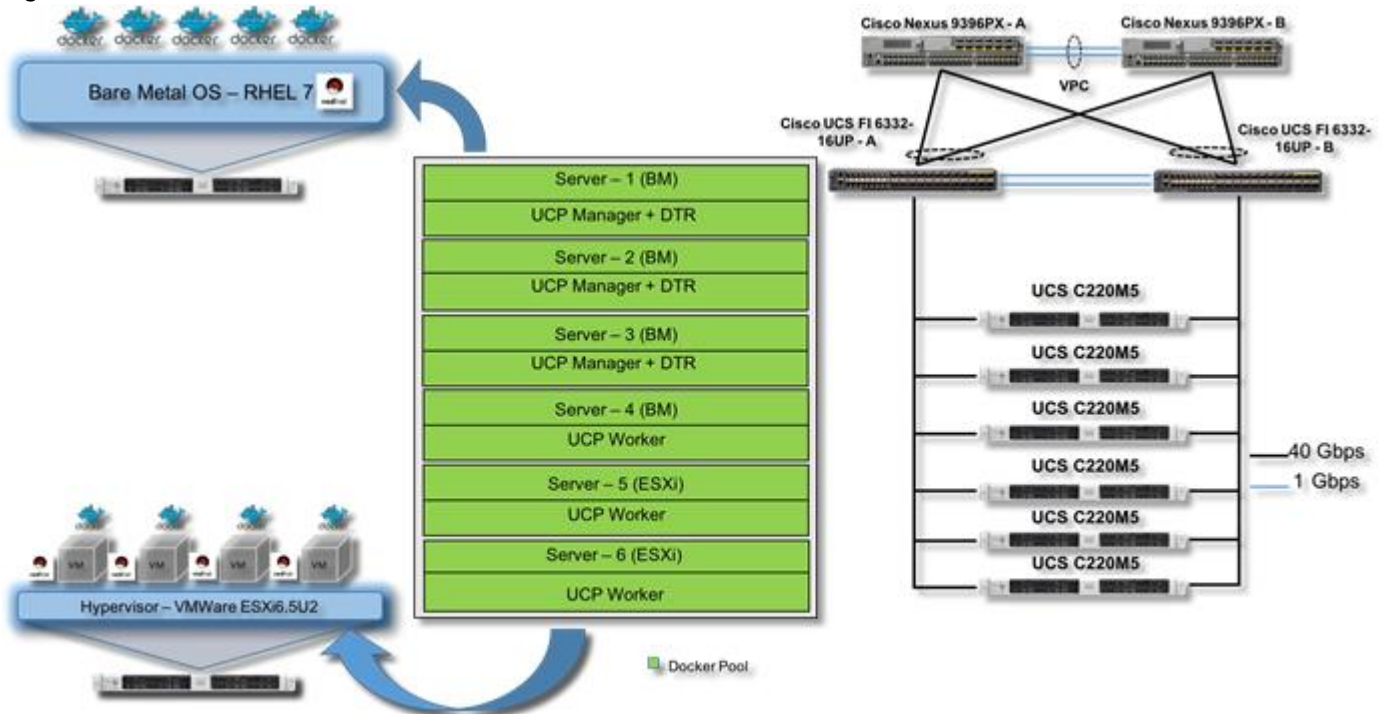


Figure 31 Mixed cluster with C-series servers - Second Architecture



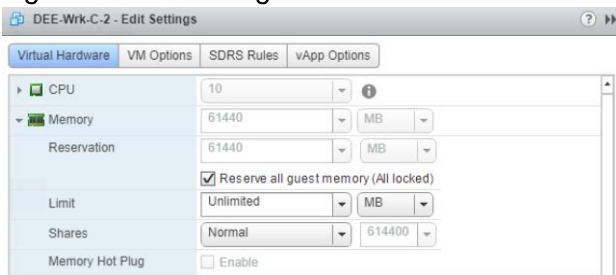
Technical Requirements for Virtual Environment

As we know the scheduling of application containers on the cluster nodes is handled through Docker UCP in swarm mode; which takes away the need for migrating VMs around the ESXi nodes. Additionally, the primary usecase of this solution is to demonstrate the statelessness of container application without storage persistency, which eliminates the need for DRS/Fault-tolerant ESXi cluster. ESXi nodes provide virtualization platform without having them to form a cluster with shared external storage for virtual machine datastore. These VMs join the existing Docker swarm mode cluster as either worker or manager nodes. Each of these VMs runs Docker Engine separately which enables them to join the Docker swarm cluster managed by Docker UCP.

Virtual Machine Sizing

For running application container workloads on the VMs, we have created four VMs per ESXi node. Each VM per ESXi node is sized with 10 vCPU and 60GB of memory footprint. In order to restrict oversubscription of resources among the VMs on an ESXi node, memory reservation is set to 100% per VM.

Figure 32 VM Sizing



Storage Considerations for Virtual Environment

All storage needs for ESXi hosts, virtual machines and Docker EE engine is being provisioned locally through the use of a specific UCS Manager Storage Profile. Local storage devices per ESXi nodes for ESXi hypervisor, virtual machine disk image and guest operating system storage have been exposed through VMWare data store units. Storage needs for Docker Engine and local containers is being exposed to each of the four VMs through RDM (Raw Device Mappings). This is essential as each VM would need a raw device for guest operating system to be configured as LVM thinpool for running Docker EE engine. Following figures show the storage configurations at the ESXi and VM level.

Figure 33 ESXi Storage Devices

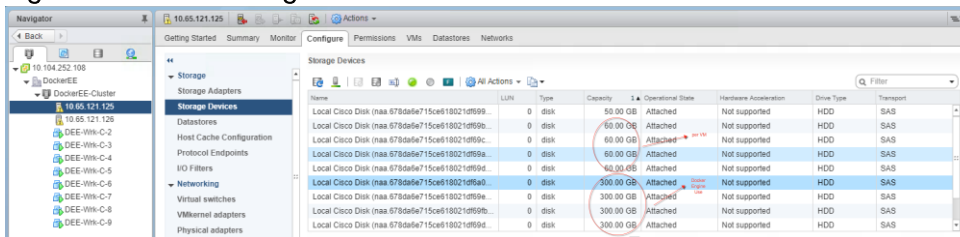


Figure 34 ESXi Datastore

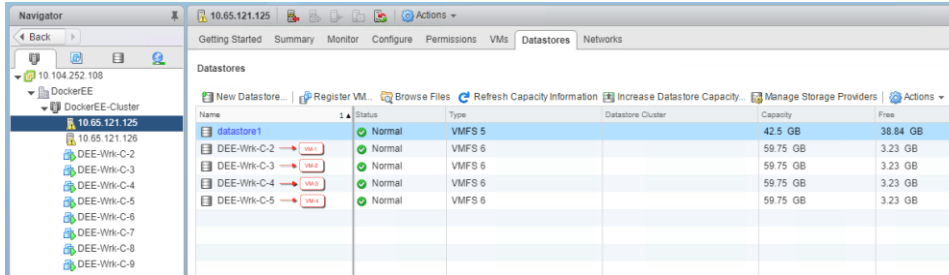
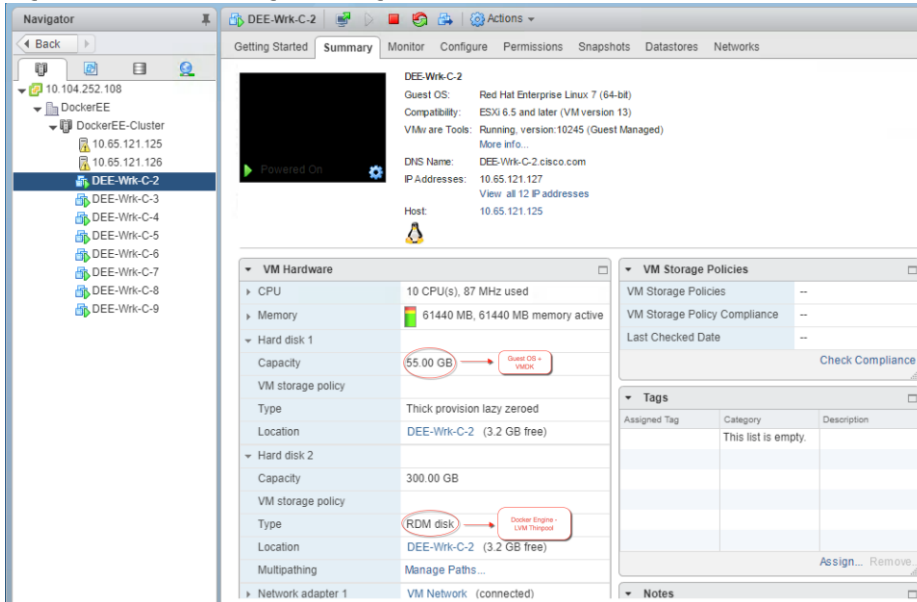


Figure 35 VM Storage Configuration



Network Considerations for Virtual Environment with Contiv

Standard VM network is used behind vSwitch to manage vmkernel interfaces and vm ports for ESXi and guest operating system host access. For Contiv container network data path, a dedicated physical network device per virtual machine is configured in passthrough mode.

Figure 36 Standard vSwitch for ESXi host/Guest OS Access and VM network

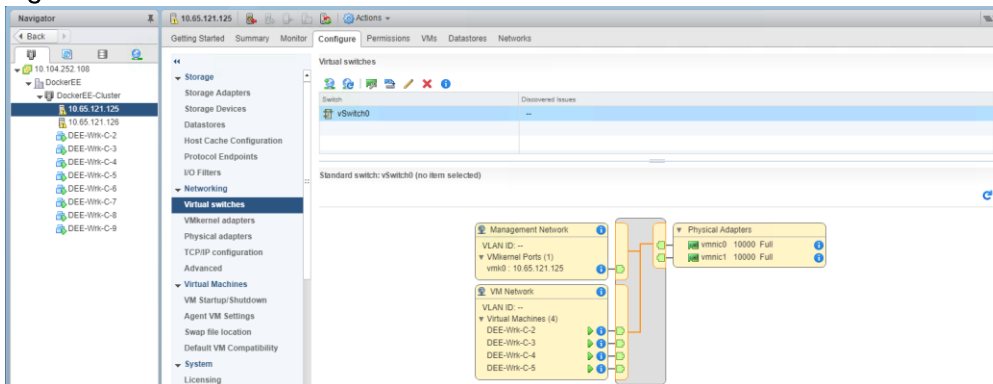
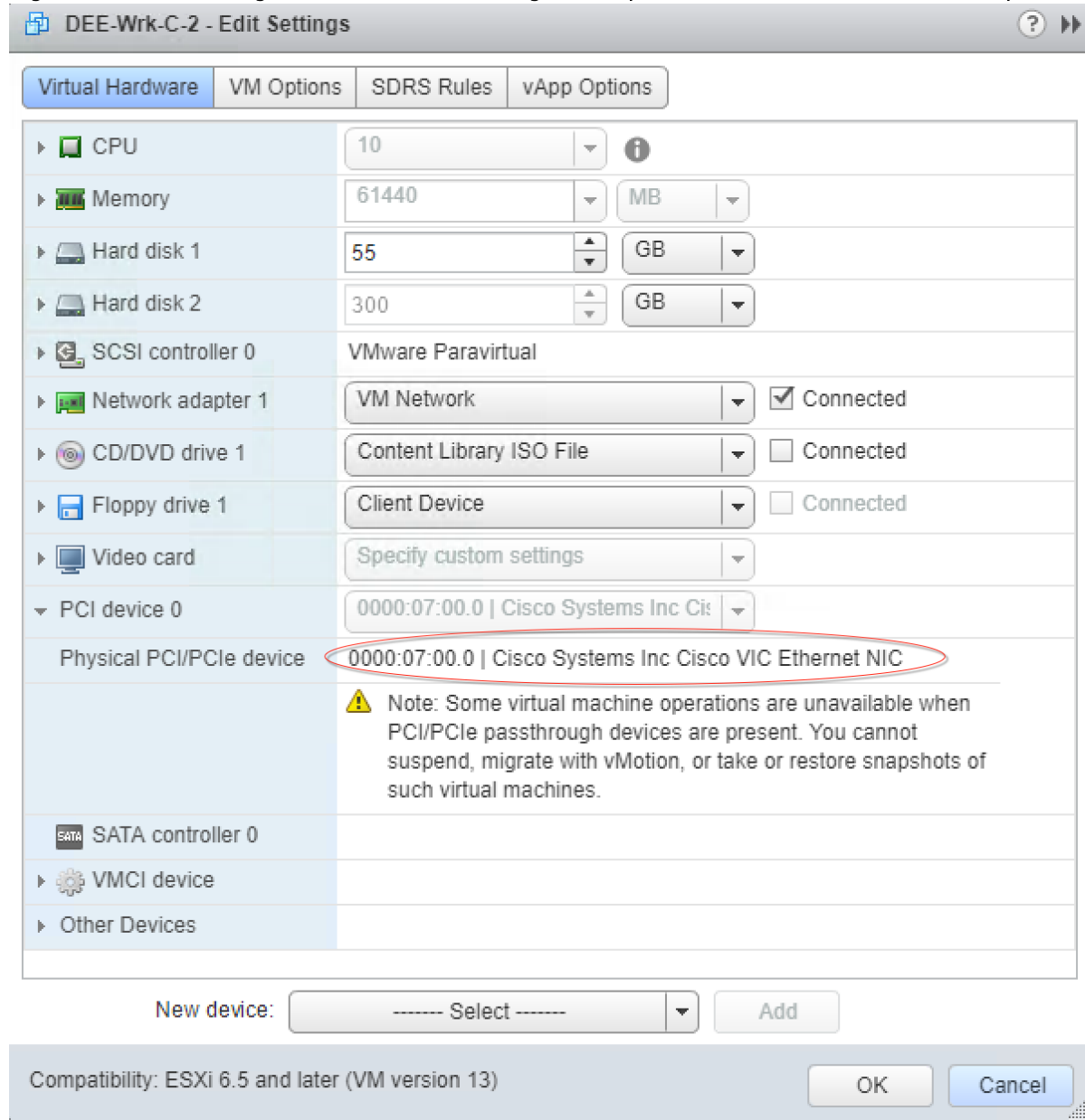


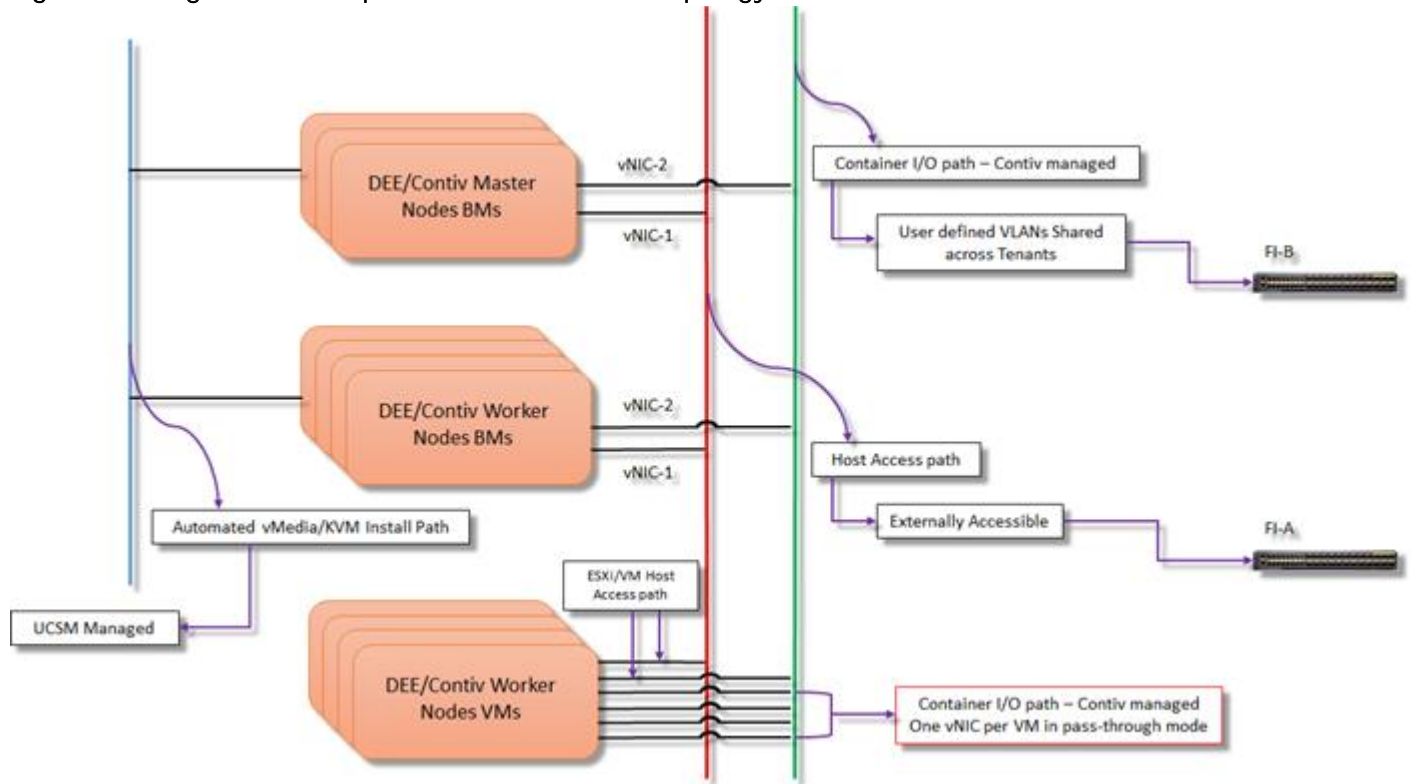
Figure 37 Passthrough network device configuration per VM for Contiv Container datapath



Logical topology

Following figures illustrate the networking configuration and logical data path.

Figure 38 Logical network paths on mixed cluster topology



Solution Deployment

This essentially comprises four main tasks to extend the existing Docker EE cluster for including additional worker nodes running on virtual machines.

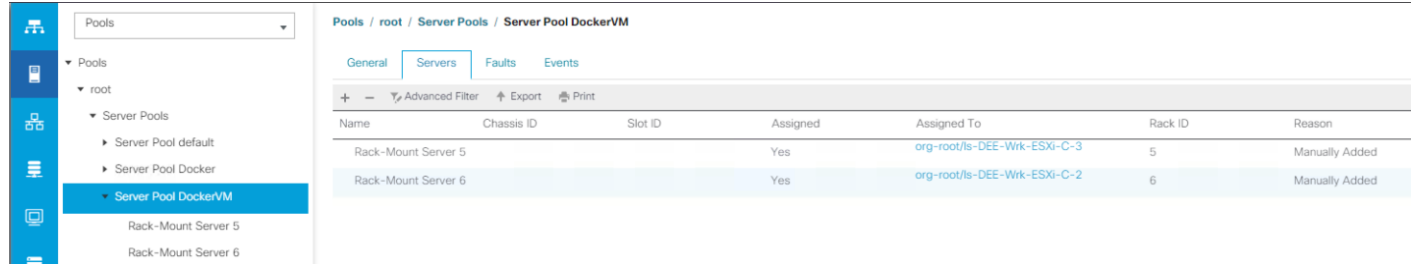
- UCS Manager configuration tasks for provisioning compute, storage and network for the two ESXi host additions
- Installing ESXi, discovering them on vCenter and deploying virtual machine
- Configuring post install OS, installing Docker EE engine and adding Docker hosts as worker nodes to the existing cluster
- Installing Contiv on newly added worker nodes to the cluster as Contiv worker nodes

UCS Manager Configuration

This solution uses manual method of deploying ESXi/Guest OS on two additional blade/rack servers for simplicity and avoiding use of additional tool sets for automating virtual infrastructure. UCS Manager configuration tasks are simple right from discovering additional blade/rack servers to associating them to the service profiles, to make them ready for the ESXi/VM/GuestOS deployment. Procedure remains the same as explained in the previous sections, except for following differences:

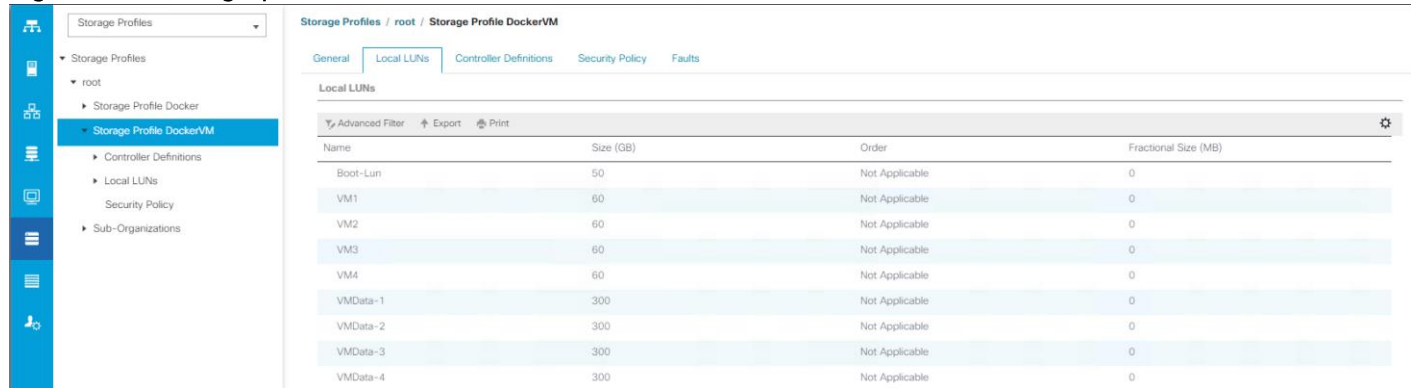
1. Create a new server pool for adding new blade/rack servers for virtual environment.

Figure 39 ESXi server pool



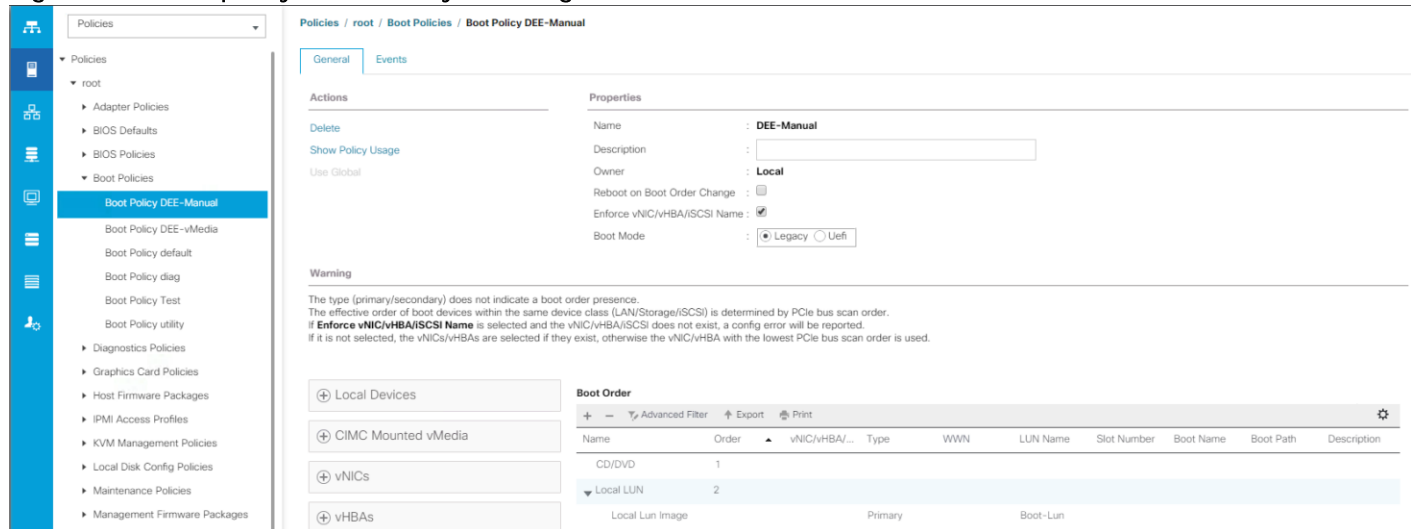
2. Create a new storage profile for ESXi hosts. Disk group policy remains the same as explained previously: RAID1 for the first architecture with blade servers and RAID10 for the second architecture with rack mount servers. Create nine LUNs as shown below. Boot-Lun is for ESXi boot, VM* Luns are for each of the VMs and corresponding guest operating systems and VMData-* are for storage for Docker EE engines running on each of the VMs.

Figure 40 Storage profile for ESXi servers



3. Create a Boot Policy with - a. CD/DVD for ESXi installation through vMedia mount and b. Local LUN with LUN name matching with that defined in the storage profile created earlier, for example - 'Boot-Lun'. Local LUN will be used as ESXi installation target and subsequent hypervisor bootup.

Figure 41 Boot policy for manually installing ESXi



- This solution uses dedicated data path per VM for Contiv to provide native fabric connectivity for the containers running on the cluster. Each ESXi node hosts four VMs, so create four vNIC templates for the Contiv use and two vNIC templates for ESXi management/VM network. A total of six vNIC templates are required. vNIC templates for Contiv should have vLANs configured for Contiv tenants. In this example, user defined vLANs are ranged from 2001 -2005 as shown.



Make sure that the vLANs for Contiv in the allowed vLAN list within the vNIC templates must have 'Native VLAN' un-checked. This is essential to avoid double tagging of the network packets exchanged between the containers, as Contiv does add appropriate vLAN IDs tags on per tenant/network basis created for its own host based network provisioning. Also, please note vNICs provisioned for Contiv will be made available to the VMs in the passthrough mode and they do not get plumbed into any other software switch at the hypervisor layer. Contiv uses this network data path for its own OVS (Open Virtual Switch).

Figure 42 vNIC template list used for ESXi nodes

Name	VLAN	Native VLAN
vNIC Template Contiv-Eth2		
vNIC Template Docker-Eth1		
vNIC Template Esxi-Eth1		
Network vlan-602	vlan-602	<input checked="" type="checkbox"/>
vNIC Template Esxi-Eth2		
Network vlan-602	vlan-602	<input checked="" type="checkbox"/>
vNIC Template VMContiv-Eth1		
Network vLAN-2001	vLAN-2001	<input type="checkbox"/>
Network vLAN-2002	vLAN-2002	<input type="checkbox"/>
Network vLAN-2003	vLAN-2003	<input type="checkbox"/>
Network vLAN-2004	vLAN-2004	<input type="checkbox"/>
Network vLAN-2005	vLAN-2005	<input type="checkbox"/>
vNIC Template VMContiv-Eth2		
Network vLAN-2001	vLAN-2001	<input type="checkbox"/>
Network vLAN-2002	vLAN-2002	<input type="checkbox"/>
Network vLAN-2003	vLAN-2003	<input type="checkbox"/>
Network vLAN-2004	vLAN-2004	<input type="checkbox"/>
Network vLAN-2005	vLAN-2005	<input type="checkbox"/>

- Create vNIC template for ESXi, do not check Fabric Failover. Make sure one vNIC is assigned to each of the Fabric A/B ids.

Figure 43 VMware ESXi management/VM Network vNIC template details

Policies / root / vNIC Templates / vNIC Template Esxi-Eth1

General | VLANs | Faults | Events | VLAN Groups

Actions

- Modify VLANs
- Modify VLAN Groups
- Delete
- Show Policy Usage
- Use Global

Properties

Name : **Esxi-Eth1**

Description :

Owner : **Local**

Fabric ID : Fabric A Fabric B Enable Failover

Redundancy

Redundancy Type : No Redundancy Primary Template Secondary Template

Target

Adapter

VM

Template Type : Initial Template Updating Template

CDN Source : vNIC Name User Defined

CDN Name :

MTU :

Policies

MAC Pool : ▼

QoS Policy : ▼

6. Create vNIC template for Contiv datapath. This is provisioned using passthrough mode to each of the VMs on the ESXi nodes.

Figure 44 vNIC template for Contiv

Policies / root / vNIC Templates / vNIC Template VMContiv-Eth2

General | VLANs | Faults | Events | VLAN Groups

Actions

- Modify VLANs
- Modify VLAN Groups
- Delete
- Show Policy Usage
- Use Global

Properties

Name : **VMContiv-Eth2**

Description :

Owner : **Local**

Fabric ID : Fabric A Fabric B Enable Failover

Redundancy

Redundancy Type : No Redundancy Primary Template Secondary Template

Target

Adapter

VM

Template Type : Initial Template Updating Template

CDN Source : vNIC Name User Defined

CDN Name :

MTU :

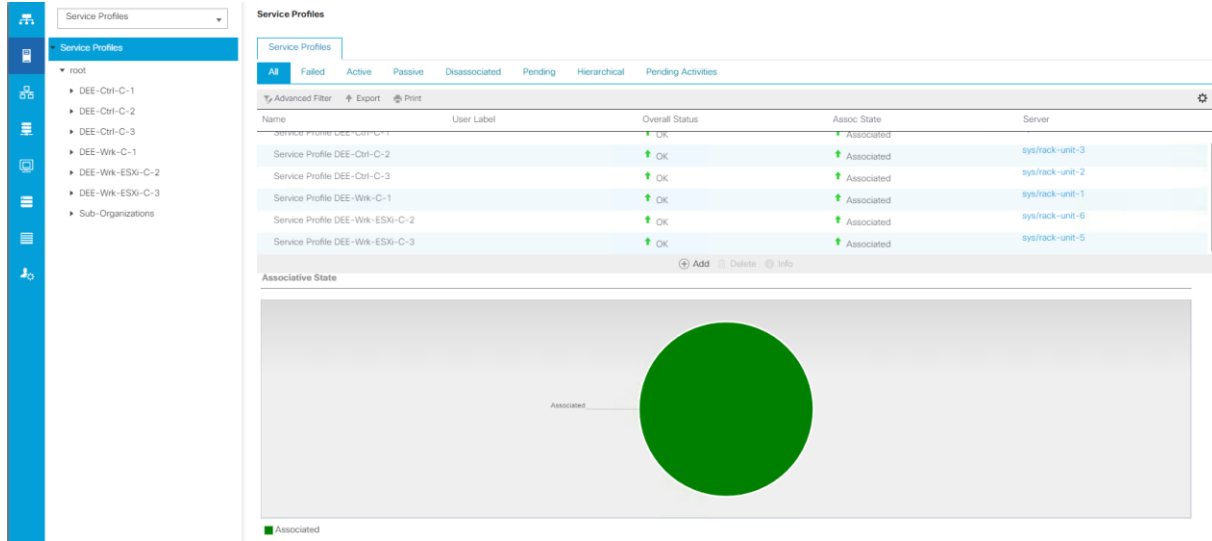
Policies

MAC Pool : ▼

QoS Policy : ▼

7. Create service profile template for ESXi nodes using storage profile, boot policy and vNIC templates as created in the previous steps and associate it with DockerVM server pool. For Contiv datapath vNICs, make sure to choose VMWarePassThrough adapter policy while creating SP template for ESXi. Rest of the policies and pools remains the same as that created for the bare metal nodes previously.
8. Instantiate two service profiles from the template created previously and associate to the servers available in the DockerVM server pool.

Figure 45 Successfully associated service profiles for mixed cluster



VMware ESXi Installation and Virtual Machine Deployment

Installation workflow for ESXi nodes follows the standard vMedia mount of the custom Cisco ISO for ESXi. Steps are listed below:

1. Reboot the server and mount the vMedia CD/DVD option with ESXi6.5 Cisco custom ISO.

Figure 46 Mounting of vMedia

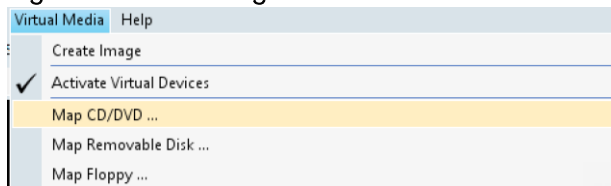
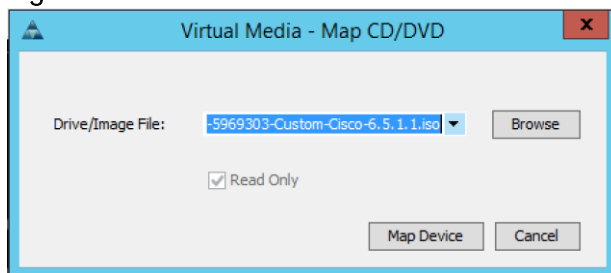
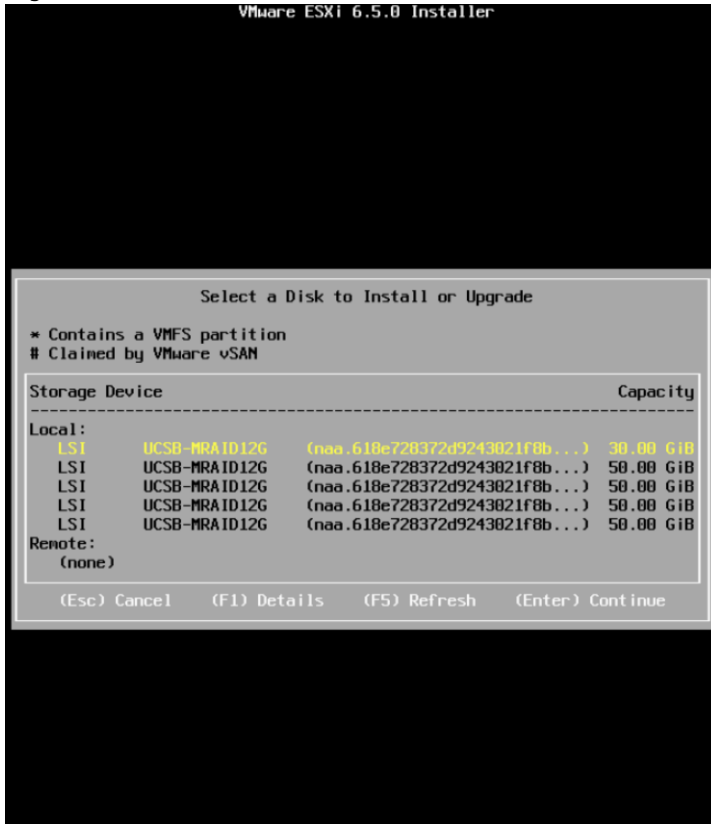


Figure 47 Cisco Custom ISO for ESXi



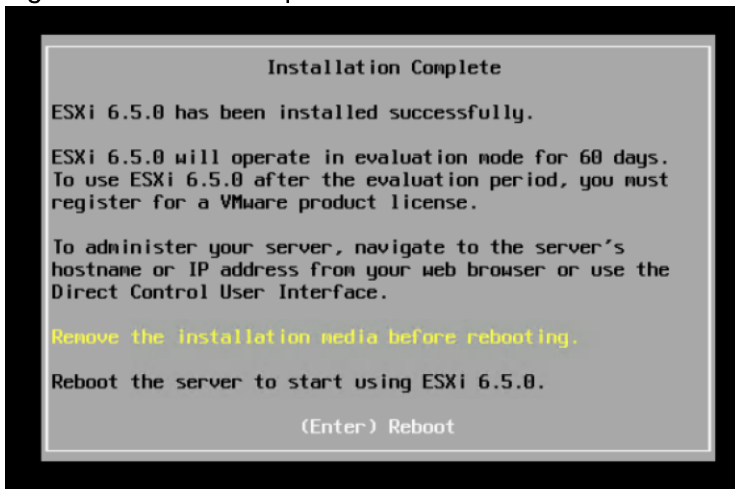
2. Select the boot lun of size 30GB which was created for ESXi install through UCS Manager Storage Profile.

Figure 48 Boot lun selection



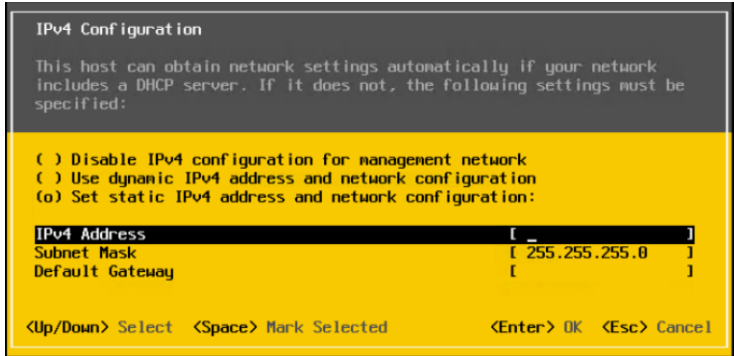
3. Press Enter to reboot after the installation is complete.

Figure 49 Install completion



4. Configure management ip address after rebooting the ESXi nodes by pressing F2 and restarting management network.

Figure 50 ESXi management network configuration



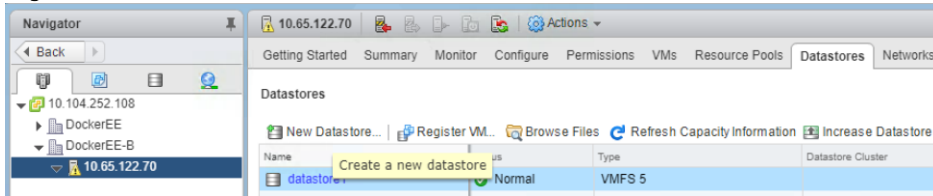
5. Add newly installed ESXi nodes to vCenter by following Getting Started link for creating datacenter and adding host to datacenter.

Figure 51 ESXi node addition



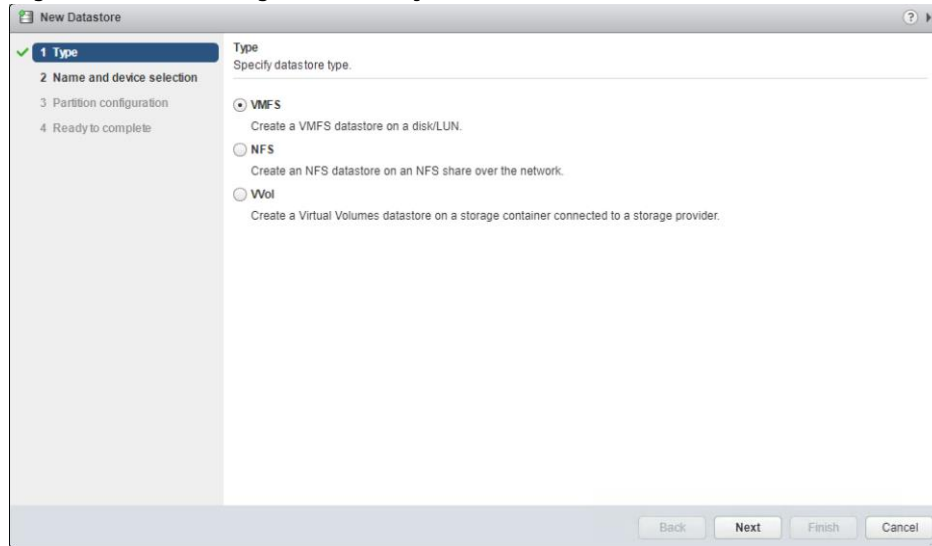
6. After adding ESXi nodes to the datacenter in vCenter, proceed with creating datastores for guest virtual machines. This datastore is needed for each VM running on the ESXi node. This will be used for guest operating system and VMDK file storage.

Figure 52 Creation of new datastore on the ESXi node



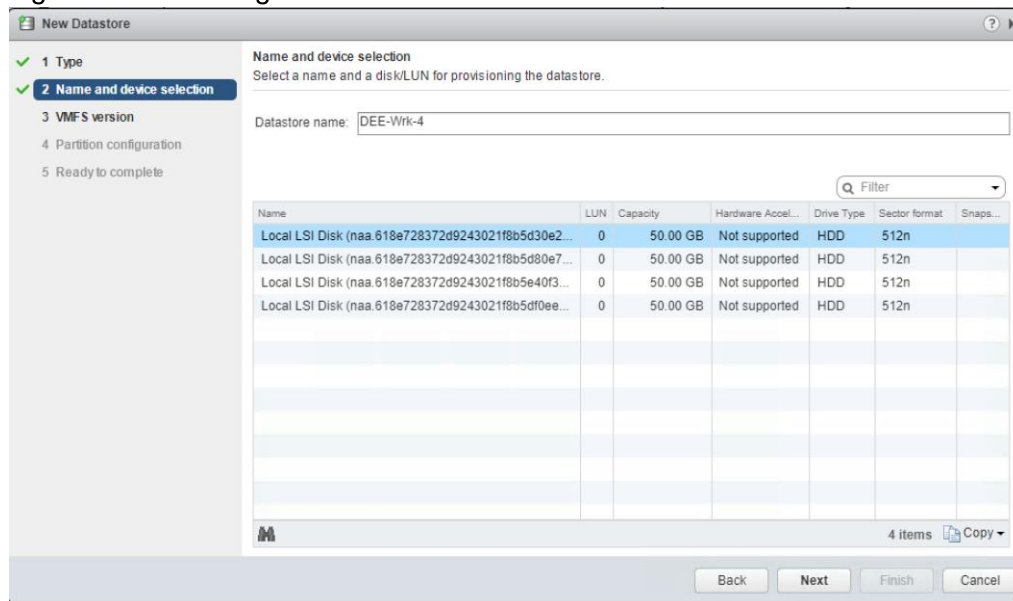
7. Select VMFS filesystem and keep the version at 6.5.

Figure 53 Selecting VMFS filesystem for the new datastore



8. Select the correct local disk. We have provisioned two LUNs for each of the ESXi nodes.

Figure 54 Selecting LUN



9. Keep rest of the settings to default options and complete datastore creation.
10. Convert additional vNICs into passthrough devices. We have provisioned vNICs for container datapath managed by Contiv for each of the virtual machines running per ESXi hosts. We have provisioned six vNICs in UCS Manager Service Profile for each of the ESXi hosts. Two vNICs are used for management and VM data path while rest of the four vNICs provisioned for each virtual machine for Contiv datapath.
11. To select PCI device option, select the Configure tab for the ESXi node, click edit and select PCI devices for passthrough.

Figure 55 Selecting PCI device option

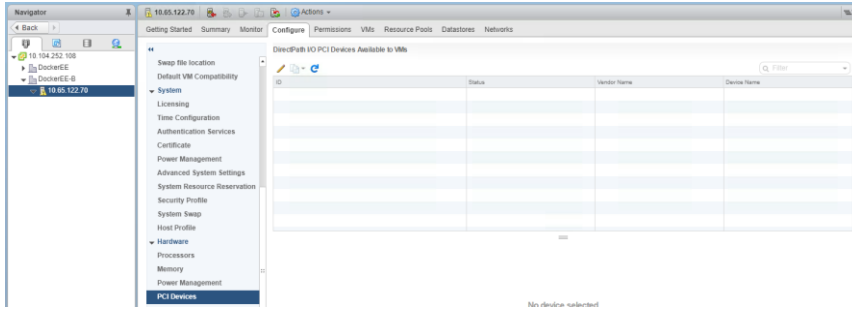
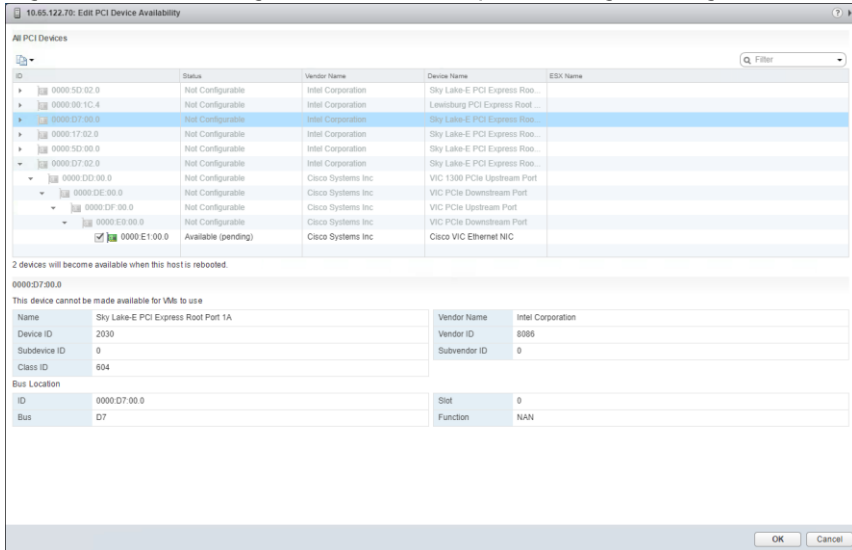
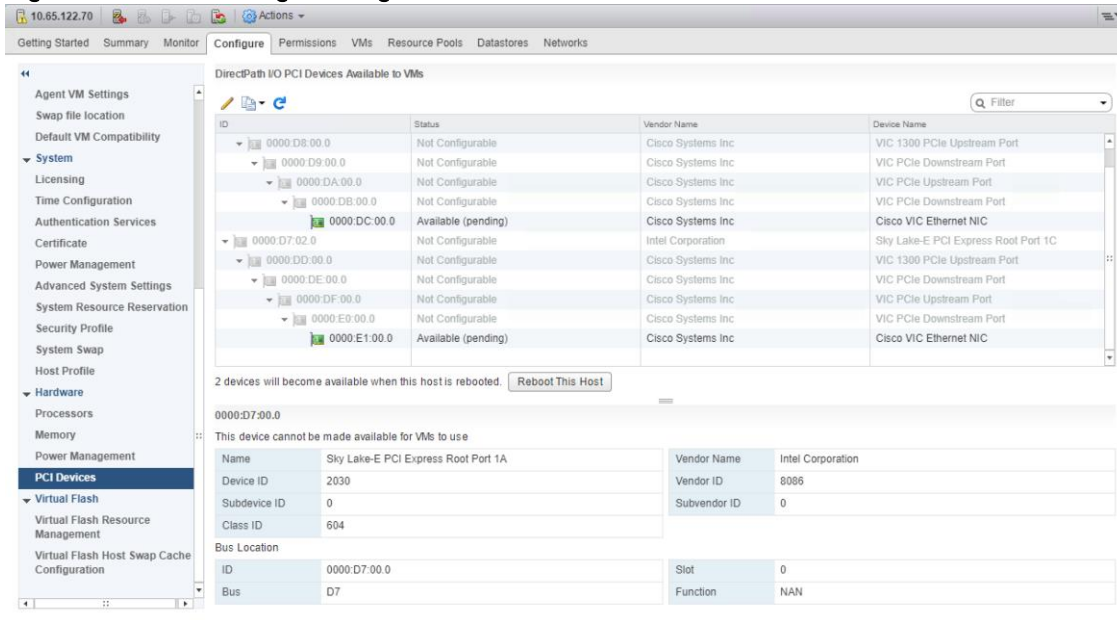


Figure 56 Selecting PCI devices for passthrough configuration



12. Reboot ESXi hosts to complete passthrough configuration.

Figure 57 Passthrough configuration



13. Deploy virtual machine by selecting the datacenter where ESXi hosts were added. Follow the standard procedure to create a virtual machine with specific configuration as needed for RDM and PCI devices as shown in the following figures.

Figure 58 Creating a new VM

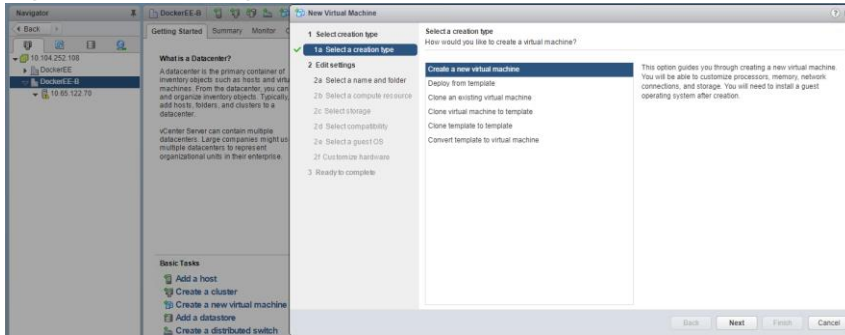


Figure 59 Selecting an appropriate datacenter

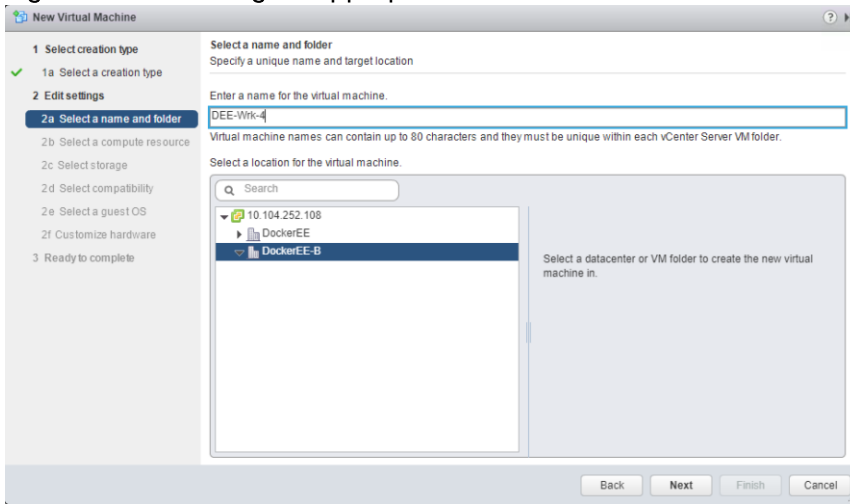


Figure 60 Selecting ESXi node within the datacenter

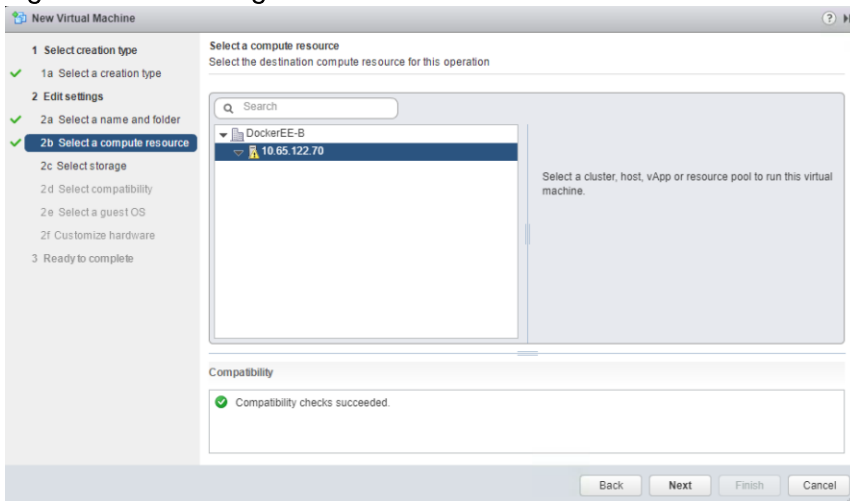


Figure 61 Selecting an appropriate datastore

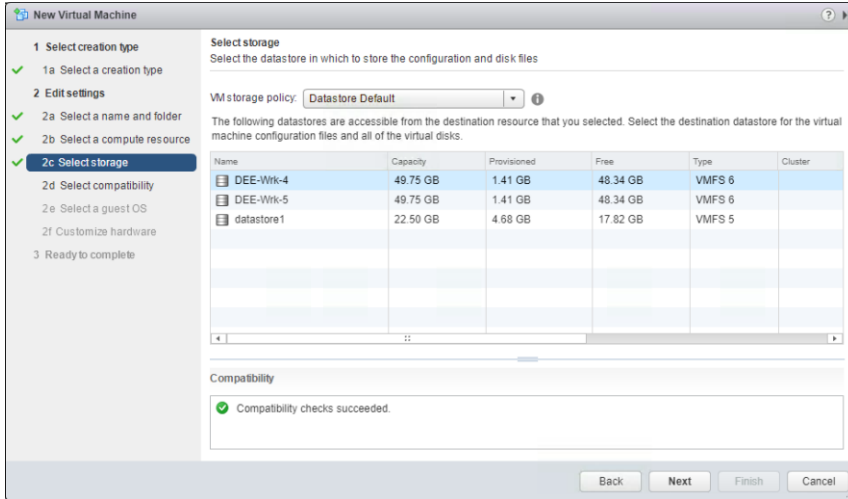


Figure 62 Selecting a guest operating system

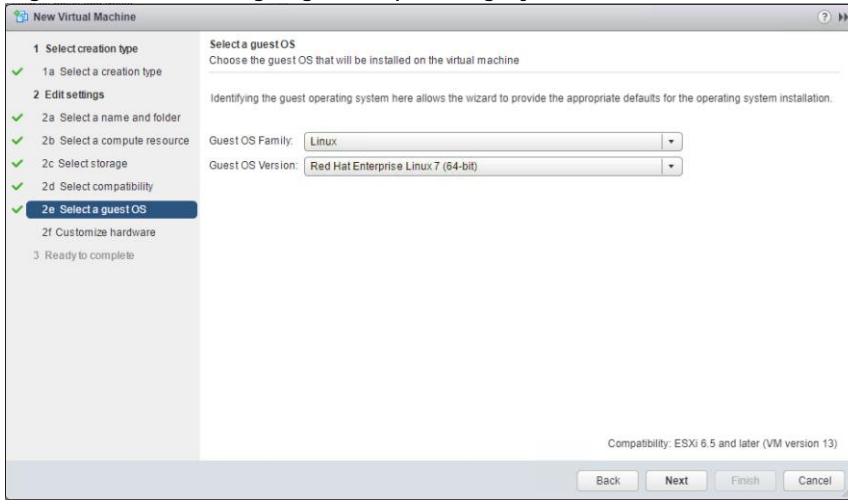


Figure 63 Customizing hardware – RDM disk for Docker Storage

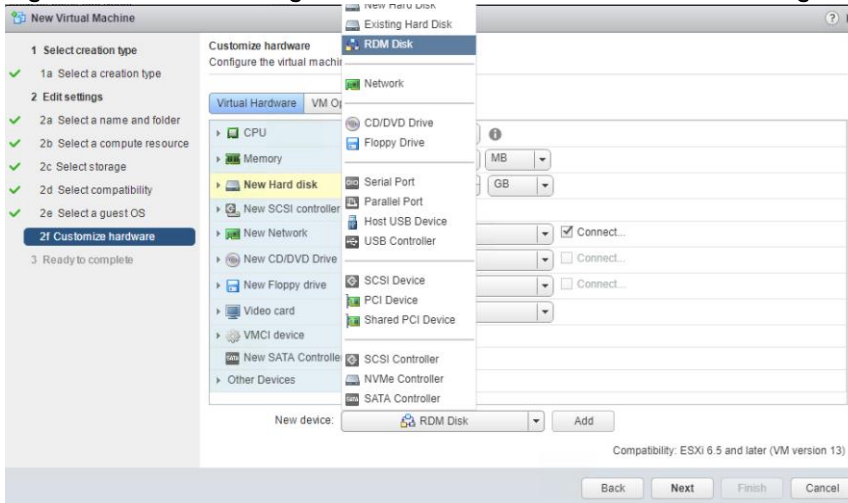


Figure 64 Selecting storage device for RDM disk to be exposed to guest OS for Docker storage

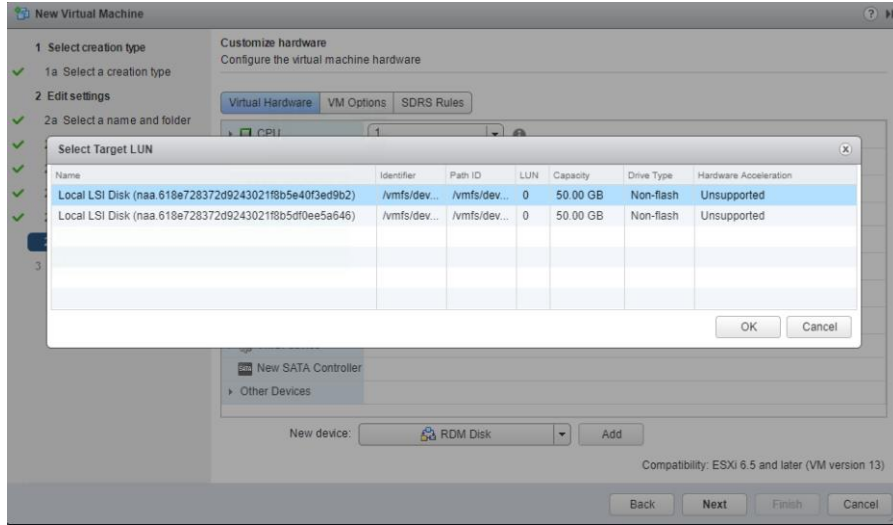


Figure 65 Selecting PCI device for Contiv datapath

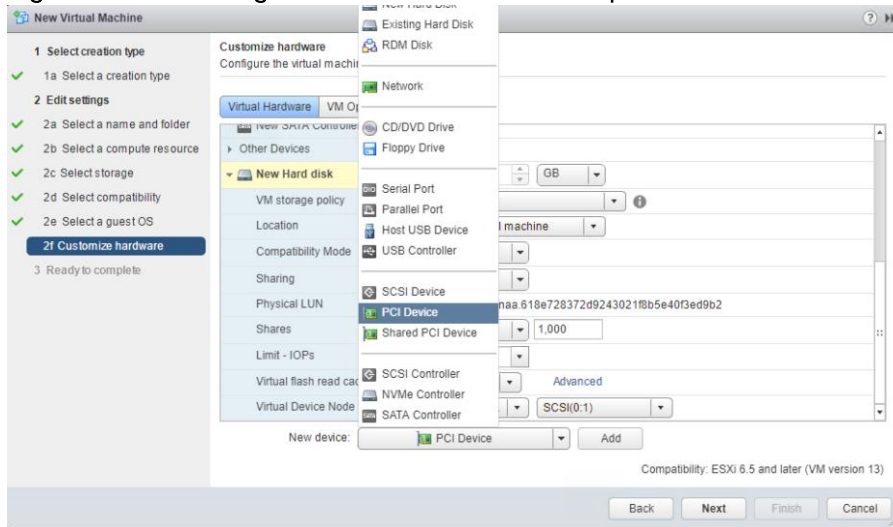


Figure 66 Choose one of the available Cisco VIC PCI device as passthrough additional vNIC for Contiv datapath

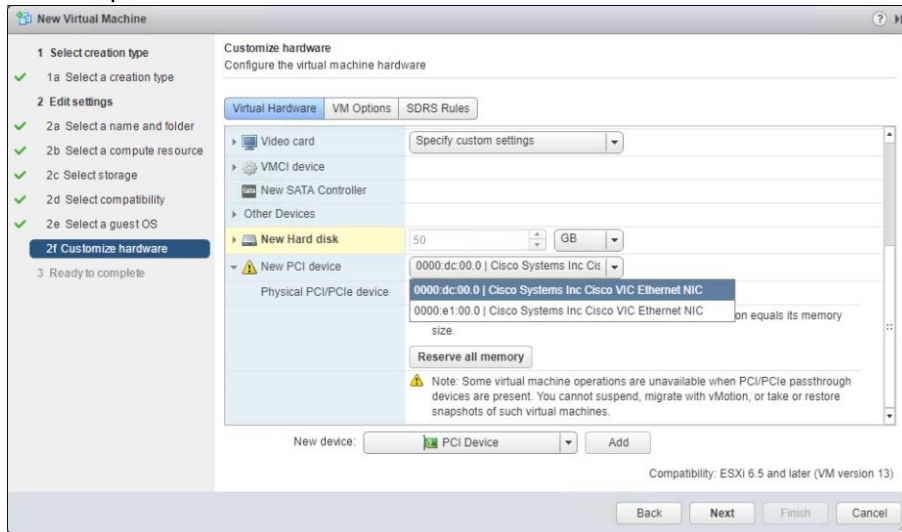
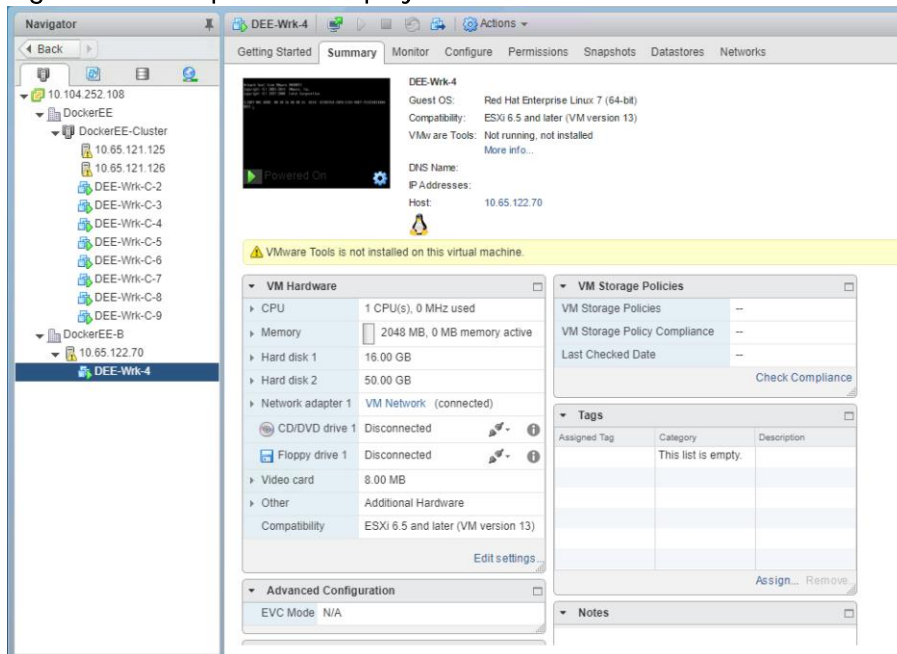


Figure 67 Complete VM deployment



14. Power on virtual machine and install OS through CD/DVD drive option by selecting ISO image from content library

Figure 68 Choosing CD/DVD drive and ISO from previously populated content library

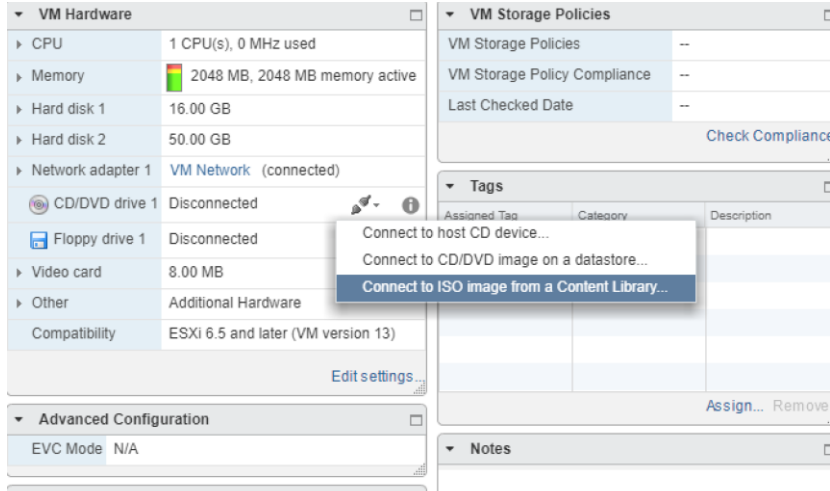
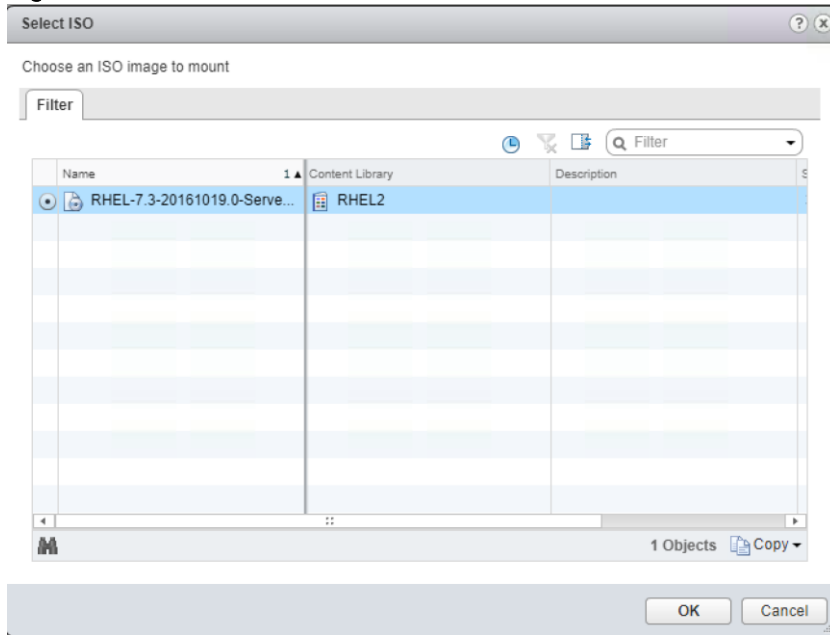
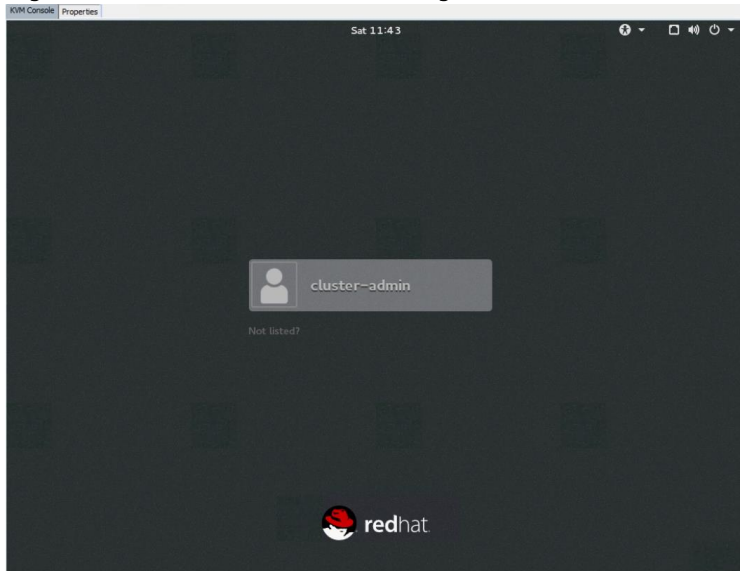


Figure 69 Select RHEL7.3 from the list to install Guest OS



15. Complete the RHEL7.3 operating system installation with standard options.

Figure 70 OS installation final stage



Docker EE Installation on Virtual Machines and Adding them to Existing UCP Cluster

The next step after guest OS installation is to complete the post install configuration and Docker EE installation. Add the newly created Docker host running on virtual machine to the existing Docker EE swarm cluster. For this, add newly provisioned virtual machines under `DEE-Nodes` and `UCP-Wrk` host group in the host inventory file used in the Ansible playbook and follow rest of the procedure as detailed in “Scaling the Cluster” section.

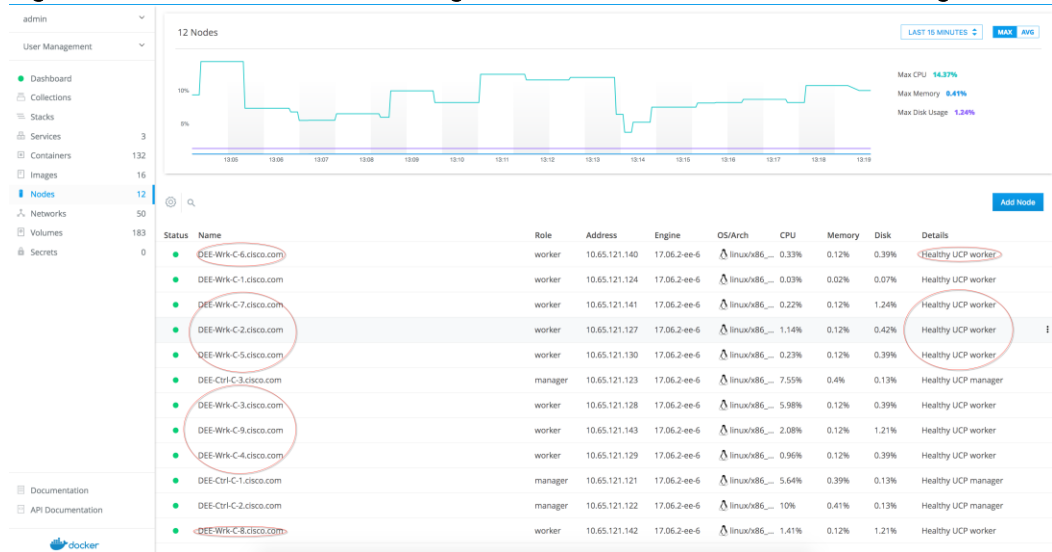
Figure 71 Ansible playbook host inventory file with new worker nodes

```
[DEE-Nodes]
DEE-Ctrl-C-1
DEE-Ctrl-C-2
DEE-Ctrl-C-3
DEE-Wrk-C-1
DEE-Wrk-C-2
DEE-Wrk-C-3
DEE-Wrk-C-4
DEE-Wrk-C-5
DEE-Wrk-C-6
DEE-Wrk-C-7
DEE-Wrk-C-8
DEE-Wrk-C-9
[UCP-Mgr]
DEE-Ctrl-C-1
[UCP-Mgr-Replicas]
DEE-Ctrl-C-2
DEE-Ctrl-C-3
[UCP-Wrk]
DEE-Wrk-C-1
DEE-Wrk-C-2
DEE-Wrk-C-3
DEE-Wrk-C-4
DEE-Wrk-C-5
DEE-Wrk-C-6
DEE-Wrk-C-7
DEE-Wrk-C-8
DEE-Wrk-C-9
```

Worker Nodes on Virtual Machines

After successful completion of the Ansible playbook for adding new nodes into the existing cluster, final state can be verified via UCP dashboard.

Figure 72 UCP dashboard showing addition of new worker nodes running on VMs



Contiv Installation on New Nodes and Adding them to Existing Contiv Cluster

Once Docker EE cluster has all the new VM worker nodes added, next step is to bring them in Conti cluster by installation Contiv with the role as Contiv worker. For this, we need to edit `cfg.yml` file inside install/ansible folder to include new nodes as worker nodes. We can find out the interface names for control/data path by logging into virtual machines and running `ifconfig -a` command. Interface with no IP address will be used for Contiv data path.

Figure 73 Sample `cfg.yml` file with newly added worker nodes

```
[root@DEE-Ctrl-C-1 ansible]# pwd
/root/contiv-1.1.7/install/ansible
[root@DEE-Ctrl-C-1 ansible]# cat cfg.yml
CONNECTION_INFO:
  10.65.121.121:
    role: master
    control: eno5
    data: eno6
  10.65.121.122:
    role: master
    control: eno5
    data: eno6
  10.65.121.123:
    role: master
    control: eno5
    data: eno6
  10.65.121.124:
    control: eno5
    data: eno6
  10.65.121.127:
    control: ens224
    data: ens192
  10.65.121.128:
    control: ens224
    data: ens192
  10.65.121.129:
    control: ens224
    data: ens192
  10.65.121.130:
    control: ens224
    data: ens192
  10.65.121.140:
    control: ens224
    data: ens192
  10.65.121.141:
    control: ens224
    data: ens192
  10.65.121.142:
    control: ens224
    data: ens192
  10.65.121.143:
    control: ens224
    data: ens192
```

Run the installation command as is using edited `cfg.yml`.

Figure 74 Contiv installation

```
PLAY RECAP *****
node1                : ok=37  changed=12  unreachable=0    failed=0
node10               : ok=22  changed=8   unreachable=0    failed=0
node11               : ok=22  changed=8   unreachable=0    failed=0
node12               : ok=22  changed=8   unreachable=0    failed=0
node2                : ok=36  changed=11  unreachable=0    failed=0
node3                : ok=36  changed=11  unreachable=0    failed=0
node4                : ok=22  changed=8   unreachable=0    failed=0
node5                : ok=22  changed=8   unreachable=0    failed=0
node6                : ok=22  changed=8   unreachable=0    failed=0
node7                : ok=22  changed=8   unreachable=0    failed=0
node8                : ok=22  changed=8   unreachable=0    failed=0
node9                : ok=22  changed=8   unreachable=0    failed=0
```

Validate successful Contiv installation by issuing `docker plugin ls` command.

Figure 75 Contiv Plugin status

```
CEE-WRK-C-2-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
31f9b69e2841 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
7a32fe13c079 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true

CEE-WRK-C-3-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
b329eb7f82b0 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
f035b433d8a8 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true

CEE-WRK-C-4-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
ae1371048a75 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true
e538a38ce572 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true

CEE-WRK-C-5-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
37a023bf9816 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
137c0ada1e458 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true

CEE-WRK-C-6-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
11f827cb1723 contiv/v2plugin:1.1.7 Contiv network plugin for Docker true
288664ab30d  docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true

CEE-WRK-C-7-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
358dc77fcoef docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
82c2d3f4982d contiv/v2plugin:1.1.7 Contiv network plugin for Docker true

CEE-WRK-C-8-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
11f6dcf897e3 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
3ed9cfa8f7dd contiv/v2plugin:1.1.7 Contiv network plugin for Docker true

CEE-WRK-C-9-1 SUCCESS | rc=0 >>
ID          NAME          DESCRIPTION          ENABLED
617ce763cad6 docker/telemetry:1.0.0.linux-x86_64-stable Docker Inc. metrics exporter true
e1f6c0e74e9  contiv/v2plugin:1.1.7 Contiv network plugin for Docker true
```

Testing and Validation for mixed cluster environment

Feature functional and high-availability test routines were performed to validate the solution and design. Tests were mainly focused on highlighting Cisco UCS, Contiv and Docker EE capabilities to manage containers and its datapath irrespective of whether they run on bare metal nodes or virtual machines. Following sample tests were validated on bare metal and virtual machine mixed cluster:

Functional Test Scenarios – Docker EE with Contiv on a Mixed Cluster Environment

1. Create network, subnet, VLAN, application profile and policy constructs
2. Create and deploy containers with Contiv network
3. Test the connectivity with all possible scenarios for L2 VLAN mode of operation under Contiv work-flow
4. Create Container-Application-Groups for Contiv network, Define Policies, Associate Policies to container-groups, ensure that the policies were enforced
5. Validate discovery of new end-points (new containers etc) on the fabric
6. While configuring various policies, networks, tenants, etc. validate the error messages and appropriate warning messages on CLI/GUI
7. Isolation policy tests and validation
8. Service discovery tests – DNS/IPAM driver basic validation

9. Container datapath and connectivity tests between different subnet via SVI/inter-vLAN routing at TOR switch

High-Availability Tests

Following set of high-availability tests were performed on the mixed cluster environment the same way as that were done on the baremetal cluster nodes. Docker EE UCP master/controllers, UCP and DTR worker nodes were subjected to the various failure scenarios and their performance was observed. Node failure was simulated by rebooting and shutting down a node and removal of that node from the chassis.

Tests Performed on Docker Enterprise Edition

1. UCP worker 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 Manager Master/Replica node reboot:
 - Tests passed. UCP management control plane through CLI/GUI was available for cluster/application container administration. UCP master node reboot did not impact container datapath and subsequent re-convergence of the cluster.
 - Same tests were performed with HAProxy external load-balancer in the mix. And UCP/DTR UI were accessible from the rest of the UCP master and DTR worker node, there were no service disruptions.
4. UCP Manager Master/Replica node shutdown:
 - Tests passed. Master/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 when the node was up.
 - Containers were deployed and the applications running on them were not disrupted on the UCP worker nodes during the unavailability of the UCP master node.
 - Same tests were performed with HAProxy external load-balancer in the mix. And UCP/DTR UI were accessible from the rest of the UCP master and DTR worker node, there were no service disruptions.
5. Docker Engine Service Restart on UCP worker 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.
8. Infrastructure component level service/process restart:

- Docker EE services runs on cluster nodes as infrastructure containers except for the Docker EE Engine, which is a **'systemd' process controlled at** the operating system level through **'systemctl' control calls**. **Docker Engine restart** tests have been covered in step 7. As part of this test case, the key services and components of the software stack on each node type came up gracefully.

Tests Performed with Contiv

1. Contiv worker node reboot/shutdown:

While container applications were running using Contiv network and policies, Contiv worker node was rebooted. Swarm scheduler restarted those application containers on the surviving Contiv worker nodes. There was no delay or failure seen. Also `docker plugin ls` showed Contiv NetPlugin was coming back to enabled state after reboot/shutdown.



When a VM worker node reboots, subsequent Docker Service deployment on a Swarm mode cluster gets significantly delayed. Follow this GitHub entry for further updates:

<https://github.com/contiv/netplugin/issues/1112>

2. Contiv master node reboot/shutdown:

Same test as above performed on the Contiv master node. Contiv master nodes run `etcd` cluster as well for the state store KV database. In a 3 node master HA cluster only one master node failure can be sustained. With one master node reboot/shutdown scenarios were tested. All the services ran normally and no delay or failure seen in moving application containers and infrastructure services on the surviving master nodes. With one master node in a failed state, new Contiv networks were successfully deployed and containers were able to consume them. Also with `etcd` cluster leader reboot/shutdown test, leadership/master role successfully migrated to one of the surviving master nodes. After the master node reboot/shutdown, when the node came up, the node was able to successfully join the cluster.

Bill of Material - Additional Components for Bare Metal and Virtual Machine Cluster

Following additional hardware/software components are needed for mixed cluster environment:

Table 14 BOM for first architecture

Component	Model	Quantity	Comments
Docker Enterprise Edition UCP Worker Nodes	B200 M5 (UCSB-B200-M5-U)	2	CPU - 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130) Memory - 12 x 16GB@2666 MHz RDIMM DIMMs - total of 192GB (UCS-MR-X16G1RS-H) Local Disks - 2 x 300 GB SAS disks for OS Boot & Docker Engine (UCS-HD300G10K12G) Network Card - 1x1340 VIC (UCSB-MLOM-40G-03) Raid Controller - Cisco MRAID 12 G

			SAS Controller (UCSB-MRAID12G)
VMWare vCenter Enterprise Plus	VMWare vCenter Enterprise Plus Subscription	1	Needed for ESXi host and vCenter

The following infrastructure components are needed for UCS C-Series second architecture:

Table 15 BOM for second architecture

Component	Model	Quantity	Comments
Docker Enterprise Edition UCP Worker Nodes	C220 M5 (UCSC-C220-M5SX)	2	CPU – 2 x Intel Xeon Gold E7 6130@2.1GHz (UCS-CPU-6130) Memory – 12 x 16GB@2666 MHz RDIMM DIMMs – total of 192GB (UCS-MR-X16G2RS-H) Local Disks – 8 x 600 GB SAS disks for OS Boot and Docker Engine (UCS-HD600G15K12N) Network Card – 1x1385 VIC (UCSC-PCIE-C40Q-03) Raid Controller – Cisco MRAID 12 G SAS Controller (UCSC-RAID-M5)
VMWare vCenter Enterprise Plus	VMWare vCenter Enterprise Plus Subscription	1	Needed for ESXi host and vCenter

Summary

The emergence of the Docker platform and the underlying support in the Linux and Windows has enabled a shift in the way that traditional applications are managed and new applications are designed and built, moving to more efficient micro services architectures. Microservices architectures are an approach to modernize and build complex applications through small, independent components that communicate with each other over language-independent APIs.

Docker Enterprise Edition on Cisco UCS platform enabled through automation tools like UCS Python SDK and Ansible provides container application platform to get deployed and managed quickly at scale. With Contiv, higher-level of networking abstraction can be achieved which secures application using a rich policy framework. A built-in service discovery and service routing for scale out services makes Contiv an indispensable component on Cisco UCS servers for container technology and microservices platform.

Cisco UCS with Contiv and Docker Enterprise Edition provides a seamless platform to run cloud native applications with containers on bare metal and virtual machine hosts homogeneously. It gives a single management control plane for administering compute, storage and network resources needed for container applications.

Invariably, Docker Enterprise Edition and Contiv on Cisco UCS infrastructure brings in a best of breed container solution for the Enterprise to run production grade application containers and microservices. While Docker EE gives us a robust container platform, Contiv provides variety of network feature sets and functionality for the applications, Cisco UCS provides a programmable infrastructure story best suited for DevOps with automation as a key feature. This solution can accelerate 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.

Appendix

Appendix – I: HAProxy Example Configuration for External Load-Balancer

HAProxy is a free, very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for very high traffic web sites and powers quite a number of the world's most visited ones. Over the years it has become the de-facto standard opensource load balancer, is now shipped with most mainstream Linux distributions, and is often deployed by default in cloud platforms.

Its software external load-balancer, which can be installed on any of the Linux host, if not selected to be installed during operating system installation. In this solution, RHEL 7 build/web-server is used for running external load balancer services for UCP, DTR and Contiv Dash boards.



Users of this guide are free to choose any of the external load-balancer of their choice, be it hardware and/or software based.

1. HAProxy is available through rhel-7-server-rpms repo. To install HAProxy –

```
# yum install haproxy
<snip>
Running transaction
  Installing : haproxy-1.5.18-3.e17_3.1.x86_64
1/1
  Verifying  : haproxy-1.5.18-3.e17_3.1.x86_64
1/1
Installed:
  haproxy.x86_64 0:1.5.18-3.e17_3.1
```

2. Edit /etc/haproxy/haproxy.cfg with below entries –

```
global
    log /dev/log      local0
    log /dev/log      local1 notice

defaults
    log             global
    mode            tcp
    option          tcplog
    option          dontlognull
    timeout connect 5000
    timeout client  50000
    timeout server  50000

### frontends
### For UCP Dash board front-end VIP Config
frontend ucp_443
    mode tcp
    bind 10.65.122.77:443
    redirect scheme https code 301 if !{ ssl_fc }
    default_backend ucp_upstream_servers
### For DTR Dash board front-end VIP Config
frontend dtr_443
    mode tcp
    bind 10.65.122.78:443
```

```

        redirect scheme https code 301 if !{ ssl_fc }
        default_backend dtr_upstream_servers_443
### For Contiv UI front-end VIP config
frontend contiv_10000
    mode tcp
    bind 10.65.122.79:443
    redirect scheme https code 301 if !{ ssl_fc }
    default_backend contiv_upstream_servers_10000
### backends
### Backend hosts(UCP Master Nodes) serving UCP Dash-board, which will be load balanced
backend ucp_upstream_servers
    mode tcp
    server DEE-Ctrl-1 10.65.122.61:443 check
    server DEE-Ctrl-2 10.65.122.62:443 check
    server DEE-Ctrl-3 10.65.122.63:443 check
### Backend hosts(DTR Nodes) serving DTR Dach-board, which will be load balanced
backend dtr_upstream_servers_443
    mode tcp
    server DEE-DTR-1 10.65.122.64:443 check
    server DEE-DTR-2 10.65.122.65:443 check
    server DEE-DTR-3 10.65.122.66:443 check
### Backend hosts(Contiv Master Nodes) serving Contiv UI, which will be load balanced
backend contiv_upstream_servers_10000
    mode tcp
    server DEE-Ctrl-1 10.65.122.61:10000 check
    server DEE-Ctrl-2 10.65.122.62:10000 check
    server DEE-Ctrl-3 10.65.122.63:10000 check

```

3. Enable and start haproxy.service -

```
# systemctl enable haproxy.service
Created symlink from /etc/systemd/system/multi-user.target.wants/haproxy.service to
/usr/lib/systemd/system/haproxy.service.
```

4. Verify haproxy.service status -

```
# systemctl status haproxy.service
haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset:
disabled)
   Active: active (running) since Sun 2017-12-03 11:28:57 IST; 5s ago
     Main PID: 5907 (haproxy-systemd)
    Memory: 2.2M
      CGroup: /system.slice/haproxy.service
              └─5907 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -
/run/haproxy.pid
              └─5909 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds
              └─5910 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy dtr_443 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy dtr_443 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy contiv_10000 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy contiv_10000 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy ucp_upstream_servers started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy ucp_upstream_servers started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy dtr_upstream_servers_443 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy dtr_upstream_servers_443 started.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy contiv_upstream_servers_10000 start-
ed.
Dec 03 11:28:57 NFS.cisco.com haproxy[5909]: Proxy contiv_upstream_servers_10000 start-
ed.
```

5. Verify by accessing the UIs via configured HAProxy URLs as shown below:

Figure 76 UCP UI accessed via HAProxy VIP (10.65.122.77):

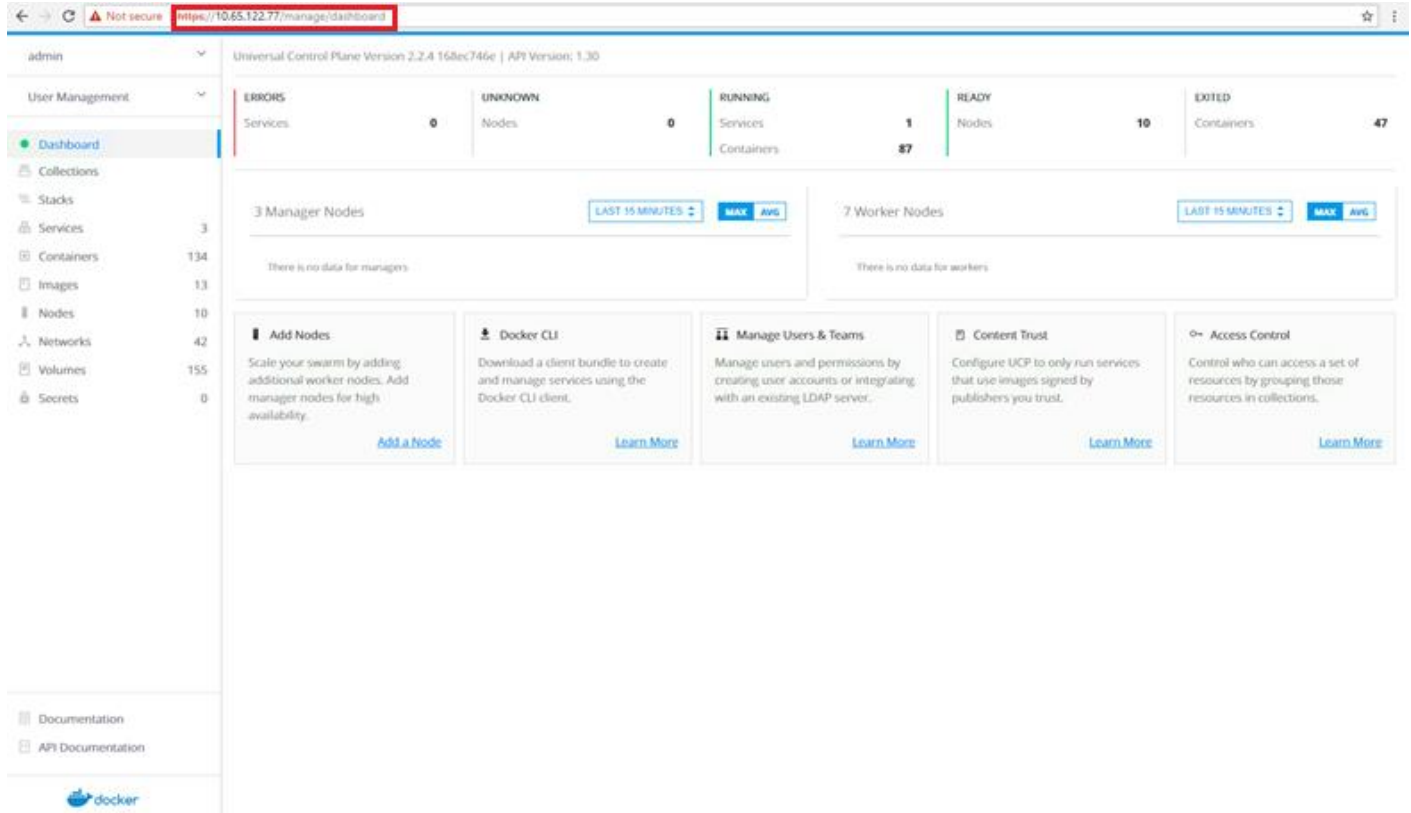


Figure 77 DTR UI accessed via HAProxy VIP (10.65.122.78):

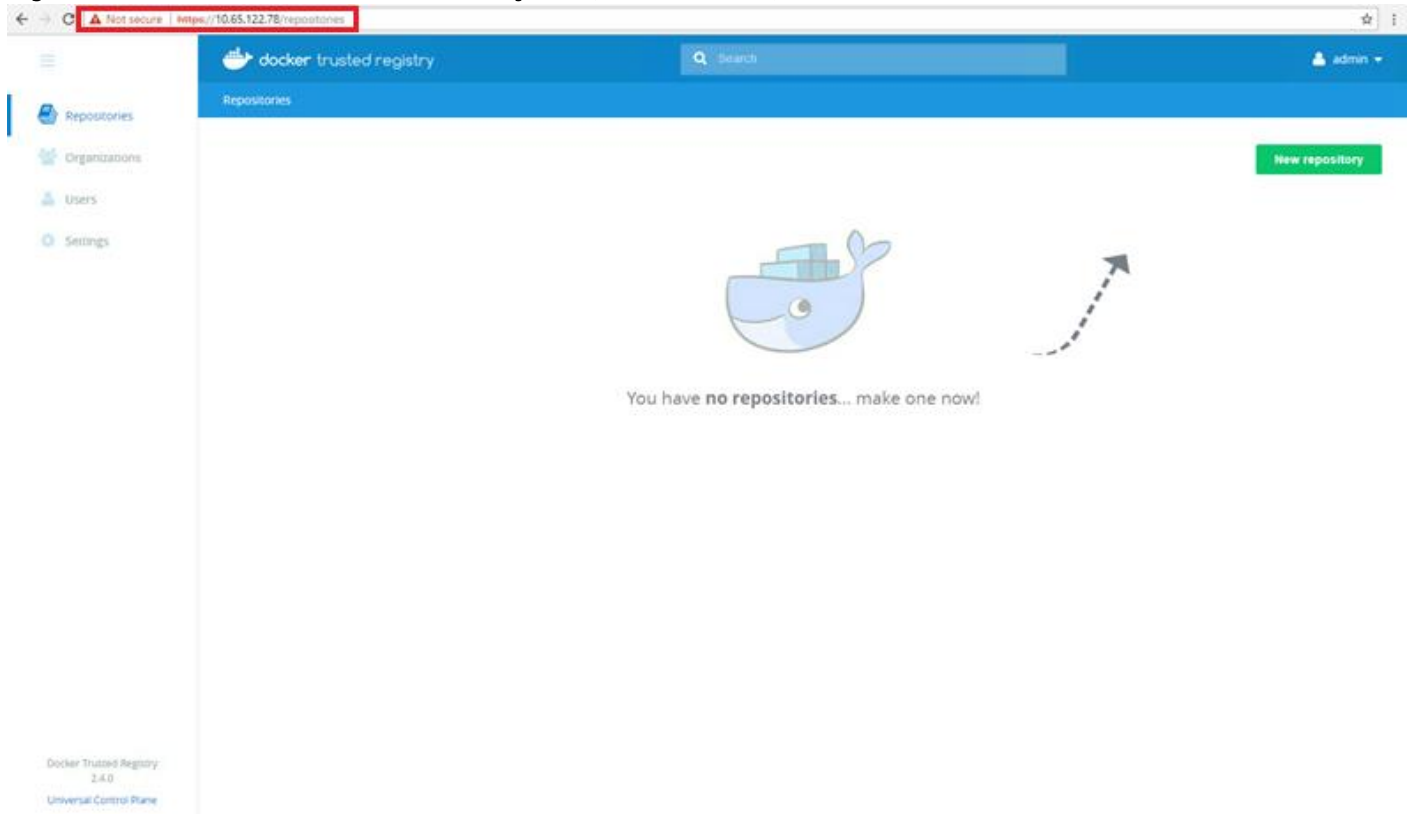
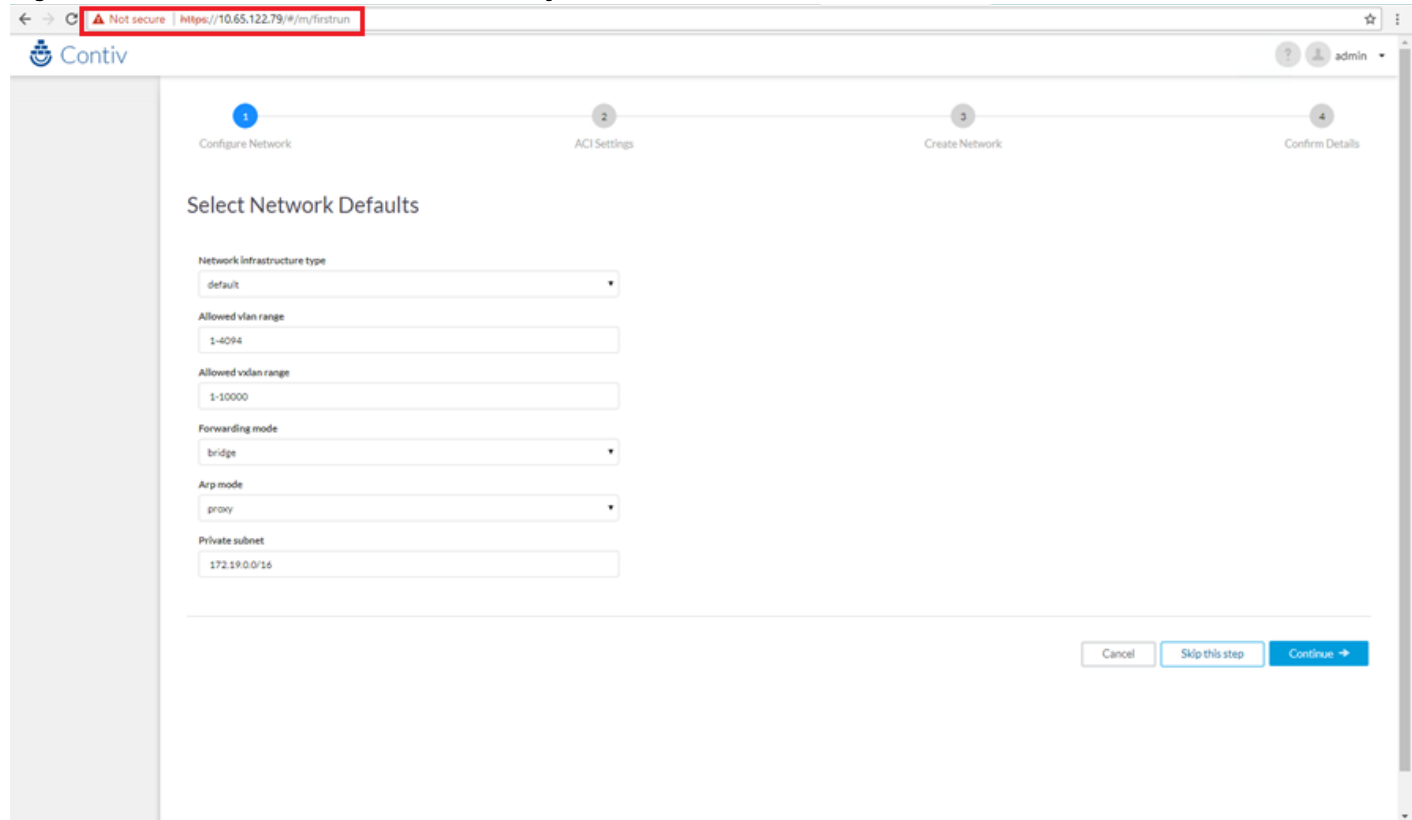


Figure 78 Contiv UI accessed via HAProxy VIP (10.65.122.79):



Appendix – 2: Ansible Playbook Host Inventory and Task Execution YAML File

Ansible Playbook Host Inventory

To execute Ansible play book tasks, host inventory file is essential. DEE-Nodes/DEE-C-Nodes are the inventory files used in our solution for both the architecture. Host grouping of the cluster nodes have been done based on the following task categories:

- Post OS install tasks including storage, firewall and Docker EE engine install are part of the **‘Common’** role.
 - This is applicable to all nodes about to form DEE cluster; hence the name common
- First UCP Manager instance configuration tasks on the first node of the cluster is managed by the **role ‘UCPswarm’**.
 - Applicable to first node in the cluster (DEE-Ctrl-1)
- UCP Manager replica configuration tasks are **managed by the role ‘UCPreplica’**.
 - Applicable to next 2 nodes in the cluster (DEE-Ctrl-2 and DEE-Ctrl-3)
- UCP Worker node configuration tasks **are managed by the role ‘UCPworker’**.
 - Applicable to all the worker nodes in the cluster (DEE-Wrk-1, DEE-Wrk-2, DEE-Wrk-3 and DEE-Wrk-4)

- DTR node configuration tasks **are managed by the role 'UCPdtr'**.
 - Applicable to first node of the DTR nodes (DEE-DTR-1)
- First DTR replica configuration tasks **managed by the role 'UCPdtr-r1'**.
 - Applicable to second DTR nodes in the cluster (DEE-DTR-2)
- Second DTR replica configuration tasks **managed by the role 'UCPdtr-r2'**.
 - Applicable to third DTR nodes in the cluster (DEE-DTR-3)

Sample host inventory file for the first architecture with 10-nodes -

```
[DEE-Nodes]
DEE-Ctrl-1
DEE-Ctrl-2
DEE-Ctrl-3
DEE-DTR-1
DEE-DTR-2
DEE-DTR-3
DEE-Wrk-1
DEE-Wrk-2
DEE-Wrk-3
DEE-Wrk-4
DEE-Wrk-5
[UCP-Mgr]
DEE-Ctrl-1
[UCP-Mgr-Replicas]
DEE-Ctrl-2
DEE-Ctrl-3
[UCP-Wrk]
DEE-DTR-1
DEE-DTR-2
DEE-DTR-3
DEE-Wrk-1
DEE-Wrk-2
DEE-Wrk-3
DEE-Wrk-4
DEE-Wrk-5
[UCP-DTR]
DEE-DTR-1
[UCP-DTR-R1]
DEE-DTR-2
[UCP-DTR-R2]
DEE-DTR-3
```

Sample host inventory file for the second architecture having 4-node cluster -

```
[DEE-Nodes]
DEE-Ctrl-C-1
DEE-Ctrl-C-2
DEE-Ctrl-C-3
DEE-Wrk-C-1
[UCP-Mgr]
DEE-Ctrl-C-1
[UCP-Mgr-Replicas]
DEE-Ctrl-C-2
DEE-Ctrl-C-3
[UCP-Wrk]
DEE-Wrk-C-1
[UCP-DTR]
DEE-Ctrl-C-1
```



```
[UCP-DTR-R1]
DEE-Ctrl-C-2
[UCP-DTR-R2]
DEE-Ctrl-C-3
```

Ansible Playbook Task Execution YAML File Configuration

Based on the host inventory files, a task execution YAML file is configured as shown below. This YAML has all the roles defined. Ansible playbook executes individual tasks as coded for each of the roles defined. This file can be used for both the architectures discussed in this solution.

Sample YAML used in this solution -

```
- hosts: DEE-Nodes
  gather_facts: no
  roles:
    - common
    - yum
    - ntp
    - firewall
    - storage
    - docker
- hosts: UCP-Mgr
  gather_facts: no
  roles:
    - UCPswarm
- hosts: UCP-Mgr-Replicas
  gather_facts: no
  roles:
    - UCPreplica
- hosts: UCP-Wrk
  gather_facts: no
  roles:
    - UCPworker
- hosts: UCP-DTR
  gather_facts: no
  roles:
    - UCPdtr
- hosts: UCP-DTR-R1
  gather_facts: no
  roles:
    - UCPdtr-r1
- hosts: UCP-DTR-R2
  gather_facts: no
  roles:
    - UCPdtr-r2
```

Appendix – 3: Contiv Data Path Troubleshooting

Contiv supports VLAN and VxLAN mode for forwarding the data traffic. VLAN/ Bridge mode is used in this solution. Contiv uses open vswitch (OVS) to provide forwarding plane for container data path. OVS kernel module is built into the base RHEL 7.3 bare metal operating system. Contiv installer installs v2plugin as a container to run OVS user space daemon (ovs-vsitchd), OVS database (ovsdb-server) and startup services. Contiv v2plugin container is part of default Docker runtime `docker-runc` services.

Verifying the Contiv global config mode

Run the command `netctl global info` to see the Contiv netctl global configuration:

```
[root@DEE-Ctrl-1 ~]# netctl global info
Fabric mode: default
Forward mode: bridge
ARP mode: proxy
Vlan Range: 1-4094
Vxlan range: 1-10000
Private subnet: 172.19.0.0/16
```

Note that the Forward mode is "bridge", which is the right mode for L2 VLAN forwarding.

Setting OVS Tool to Query the OVS Datapath

By default, the Contiv cluster nodes does not have the ovs tool set (for example, ovs-vsctl, ovsdb-tool and so on) installed. The OVS tool set is built into the v2plugin container. For Contiv OVS bridge inspection and finding the OVS datapath, login to the v2plugin container and run OVS tool set within the container. For this identify the v2plugin container first. All the third party plugins reside at `'/var/lib/docker/plugin'` directory on the cluster nodes. To see the ovs-vsctl within in the v2plugin container, run:

```
[root@DEE-Ctrl-1 ~]# cd /var/lib/docker/plugins/
[root@DEE-Ctrl-1 plugins]# ls -ltr
total 0
drwx-----. 3 root root 19 Dec 12 20:20 storage
drwx-----. 4 root root 63 Dec 12 20:20
f9d57a0003e54ece3372342c0e7aa0a0f2a2c60225049b15cc7317265b7243a8
drwx-----. 2 root root 6 Dec 13 21:27 tmp
drwx-----. 3 root root 39 Dec 13 21:27
fce04ff433c14d188af992ec3481137ac33aea027ba4e7313d81d18721f24da6
# docker plugin ls
ID                NAME                                     DESCRIPTION
ENABLED
f9d57a0003e5      docker/telemetry:1.0.0.linux-x86_64-stable  Docker Inc. metrics ex-
porter           true
fce04ff433c1      contiv/v2plugin:1.1.7                       Contiv network plugin for
Docker           true
```



First 12 charactres of container ID matches with Contiv plugin directory inside the `/var/lib/docker/plugins` directory and this is the Contiv v2plugins container ID.

```
docker-runc exec fce04ff433c14d188af992ec3481137ac33aea027ba4e7313d81d18721f24da6 ovs-vsctl
show
```

For example:

```
[root@DEE-Ctrl-1 plugins]# docker-runc exec
fce04ff433c14d188af992ec3481137ac33aea027ba4e7313d81d18721f24da6 ovs-vsctl show
d84f36b2-eb43-4f79-b600-bea009d77aa0
Manager "ptcp:6640"
Bridge contivVxlanBridge
Controller "tcp:127.0.0.1:6633"
is_connected: true
fail_mode: secure
Port "vxif106512269"
Interface "vxif106512269"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.69", tos=inherit}
port "vxif106512263"
Interface "vxif106512263"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.63", tos=inherit}
Port "vxif106512262"
```

```

    Interface "vxif106512262"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.62", tos=inherit}
Port "vxif106512267"
Interface "vxif106512267"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.67", tos=inherit}
Port "vxif106512265"
Interface "vxif106512265"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.65", tos=inherit}
Port "vxif106512264"
Interface "vxif106512264"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.64", tos=inherit}
Port "vxif106512270"
Interface "vxif106512270"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.70", tos=inherit}
Port "vxif106512266"
Interface "vxif106512266"
type: vxlan
    options: {dst_port="8472", key=flow, remote_ip="10.65.122.66", tos=inherit}
Port "vxif106512268"
Interface "vxif106512268"
type: vxlan
options: {dst_port="8472", key=flow, remote_ip="10.65.122.68", tos=inherit}
Port "contivh0"
tag: 2
Interface "contivh0"
type: internal
Bridge contivVlanBridge
Controller "tcp:127.0.0.1:6634"
is_connected: true
fail_mode: secure
Port "eno6"
Interface "eno6"

```

Contiv OVS Bridge

Contiv relies on two main OVS bridges: contivVlanBridge and contivVxlanBridge.

Contiv will add OVS port on contivVlanBridge when you create container with VLAN network. Similarly, when you create container with VxLAN network, Contiv will add OVS port on contivVxlanBridge.

```

#docker-runc exec fce04ff433c14d188af992ec3481137ac33aea027ba4e7313d81d18721f24da6 ovs-
vsctl show
d84f36b2-eb43-4f79-b600-bea009d77aa0
  Manager "ptcp:6640"
  Bridge contivVxlanBridge
    <snip>
  Bridge contivVlanBridge
    Controller "tcp:127.0.0.1:6634"
      is_connected: true
      fail_mode: secure
      Port "eno6"
        Interface "eno6"
      Port "vvport1"
        tag: 1001
        Interface "vvport1"

```

In the above example, note that the vlan tag is 1001; this is the vlan used for contiv network for deploying containers.

The port/ interface is eno6, which is the physical port assigned for container datapath. Interface vvport1 is the virtual port which pairs up with the virtual Ethernet (veth), which the containers use for network connectivity.

When you have multiple containers, you will find multiple vvport interfaces shown under the same contivVlanBridge.

For example:

```
Bridge contivVlanBridge
  Controller "tcp:127.0.0.1:6634"
    is_connected: true
  fail_mode: secure
  Port "eno6"
    Interface "eno6"
  Port "vvport2"
    tag: 1001
    Interface "vvport2"
  Port "vvport1"
    tag: 1001
    Interface "vvport1"

[root@DEE-Ctrl-1 plugins]# docker ps|grep alpine
753a6c59c74f      alpine:latest      "sleep 10000"      3 minutes
ago            Up 3
minutes
      alpine-1.15.t0o2k9s510zlv2n7f9hezlq36
5cb287578752      alpine:latest      "sleep 10000"      2 hours
ago            Up 2
hours
      alpine-1.1.pb9w2mkodi912ps1xcmmr0144
```

Correlating OVS Ports with Container Ports

For identifying the vvport connection with its respective container, follow the steps to get the interface id of the host and OVS port:

To identify the veth on the container, run 'ip a' inside the running container as shown below:

```
[root@DEE-Ctrl-1 plugins]# docker exec -it 753a6c59c74f ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
122: eth0@if121: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue state UP
    link/ether 02:02:64:64:64:09 brd ff:ff:ff:ff:ff:ff
    inet 100.100.100.9/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet 100.100.100.1/32 scope global eth0
        valid_lft forever preferred_lft forever
```

Now run the 'ip a' command on one of the cluster nodes where the containers are running on Contiv network.

```
[root@DEE-Ctrl-1 plugins]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```

inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: eno5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:25:b5:99:99:8f brd ff:ff:ff:ff:ff:ff
    inet 10.65.122.61/24 brd 10.65.122.255 scope global eno5
        valid_lft forever preferred_lft forever
    inet6 fe80::225:b5ff:fe99:998f/64 scope link
        valid_lft forever preferred_lft forever
3: eno6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq master ovs-system state UP
    qlen 1000
    link/ether 00:25:b5:99:99:9f brd ff:ff:ff:ff:ff:ff

<snip>

119: vviewport1@if120: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-
system state UP
    link/ether 7a:2a:43:65:8b:82 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::782a:43ff:fe65:8b82/64 scope link
        valid_lft forever preferred_lft forever
121: vviewport2@if122: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-
system state UP
    link/ether 66:d8:74:d9:f3:b1 brd ff:ff:ff:ff:ff:ff link-netnsid 5
    inet6 fe80::64d8:74ff:fed9:f3b1/64 scope link
        valid_lft forever preferred_lft forever

```

From the above example, the correlation between container veth and vviewport on the OVS bridge managed by Contiv can be seen.

Dumping OVS Flow

Contiv supports a policy which internally translates into OVS flows. You can dump the flow tables with the following command (keep in mind you have to use OPENFLOW13):

```

[root@DEE-Ctrl-1 plugins]# docker-runc exec
fce04ff433c14d188af992ec3481137ac33aea027ba4e7313d81d18721f24da6 ovs-ofctl dump-flows con-
tivVlanBridge -O OPENFLOW13
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x20, duration=407005.286s, table=0, n_packets=0, n_bytes=0, priori-
ty=102,udp,in_port=1,tp_dst=53 actions=goto_table:1
cookie=0x1e, duration=407006.286s, table=0, n_packets=0, n_bytes=0, priori-
ty=101,udp,dl_vlan=4093,dl_src=02:02:00:00:00:00/ff:ff:00:00:00:00,tp_dst=53 ac-
tions=pop_vlan,goto_table:1
cookie=0x1c, duration=407006.286s, table=0, n_packets=12282, n_bytes=783768, priori-
ty=100,arp,arp_op=1 actions=CONTROLLER:65535
cookie=0x1d, duration=407006.286s, table=0, n_packets=0, n_bytes=0, priori-
ty=100,udp,dl_src=02:02:00:00:00:00/ff:ff:00:00:00:00,tp_dst=53 actions=CONTROLLER:65535
cookie=0x1a, duration=407006.286s, table=0, n_packets=0, n_bytes=0, priority=1 ac-
tions=goto_table:1
cookie=0x21, duration=407005.286s, table=1, n_packets=0, n_bytes=0, priority=100,in_port=1
actions=goto_table:6
cookie=0x2b, duration=7252.292s, table=1, n_packets=0, n_bytes=0, priority=10,in_port=2
actions=write_metadata:0x100000000/0xff00000000,goto_table:3
cookie=0x32, duration=1942.451s, table=1, n_packets=0, n_bytes=0, priority=10,in_port=3
actions=write_metadata:0x100000000/0xff00000000,goto_table:3
cookie=0x1b, duration=407006.286s, table=1, n_packets=0, n_bytes=0, priority=1 ac-
tions=goto_table:4
cookie=0x19, duration=407006.286s, table=3, n_packets=0, n_bytes=0, priority=1 ac-
tions=goto_table:4

```

```

cookie=0x2c, duration=7252.292s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.2 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x2d, duration=7208.685s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.5 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x2e, duration=7208.586s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.3 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x2f, duration=7207.043s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.4 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x30, duration=1943.191s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.13 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x31, duration=1942.758s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.7 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x33, duration=1942.451s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.9 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x34, duration=1942.322s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.14 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x35, duration=1914.977s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.10 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x36, duration=1914.459s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.15 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x37, duration=1913.657s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.8 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x38, duration=1912.201s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.11 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x39, duration=1912.132s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.12 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x3a, duration=1910.787s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.16 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x3b, duration=1910.205s, table=4, n_packets=0, n_bytes=0, priority=100,ip,metadata=0x100000000/0xff0000000,nw_dst=100.100.100.6 actions=write_metadata:0/0xffff,goto_table:5
cookie=0x17, duration=407006.286s, table=4, n_packets=0, n_bytes=0, priority=1 actions=goto_table:5
cookie=0x18, duration=407006.286s, table=5, n_packets=0, n_bytes=0, priority=1 actions=goto_table:6
cookie=0x16, duration=407006.286s, table=6, n_packets=0, n_bytes=0, priority=1 actions=goto_table:9
cookie=0x1f, duration=407006.286s, table=9, n_packets=0, n_bytes=0, priority=1 actions=NORMAL

```

Contiv Handling ARP Requests

Contiv handles ARP request through ARP proxy mechanism.

Before understanding how Contiv deals with ARP request, it is important to know Contiv has an internal in-memory database which stores all the endpoint (container) information such as IP, MAC, location and so on. That is, whenever user boot up a container, Contiv will automatically add the endpoint information to the in-memory database.

Use case #1: Container send ARP request to another container under same VLAN. (Proxy ARP mode)

Suppose container1 want to send a ping packet to container2 for the first time, then:

- Container will send an ARP request out to OVS contivVlanBridge
- Since the system is running Proxy ARP mode, OVS will follow the packet to the Contiv OVS controller
- OVS controller will look up the in-memory db
 - If the destination is in the same VLAN, controller will reply the ARP regardless the target container is in the same host or not
 - If the destination is not found, controller will forward the ARP request to uplink and let the upstream device to deal with the arp request

Use case #2: Container send ARP request to another container under same VLAN (Flood mode)

Suppose container1 wants to send a ping packet to container2 for the first time, then:

- Container will send an ARP request to the OVS contivVlanBridge
- Since the system is not running Proxy ARP mode, OVS will broadcast the ARP request to all other host; it will work like regular ARP request

About the Authors

Rajesh Kharya, Technical Marketing Engineering, 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 bare metal 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.

Sindhu Sudhir, Technical Marketing Engineering, Cisco UCS Solutions Engineering, Cisco Systems Inc.

Sindhu Sudhir is part of Cisco UCS Solutions Engineering team. In her current role she is focusing on Container technologies and infrastructure automation on Cisco UCS platform.

Acknowledgements

- Cisco Systems: Ka Hou Lei, Vishwanath Jakka, Babu Mahadevan, Meenakshi Kaushik, John Day
- Docker: Uday Shetty, Bradley Wong