Ubuntu OpenStack Architecture on Cisco UCS Platform
Last Updated: June 25, 2014
About the Authors

Mehul Bhatt, Virtualization Architect, Server Access Virtualization Business Unit (SAV BU), Cisco Systems

Mehul Bhatt has over 12 years of experience in virtually all layers of computer networking. His focus area includes Unified Compute Systems, network and server virtualization design. Prior to joining Cisco Technical Marketing team, Mehul was Technical Lead at Cisco, Nuova systems and Bluecoat systems. Mehul holds a Master’s degree in computer systems engineering and holds various Cisco career certifications.
Acknowledgment

For the support and contribution to the design, validation, and creation of this Cisco Validated Design, I would like to thank:

- Rakhee Gorthi - Cisco
- Gopi Krishna S - Cisco
- Robert Stammer - Cisco
- Samantha Jiyan - Canonical
- Ashok Rajgopalan - Cisco
- Mike Andren - Cisco
- Chris Ricker - Cisco
- Mark Voelker - Cisco
- Sindhu Sudhir - Cisco
- Sankar Jayaram - Cisco
About 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.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB’s public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

Cisco and the Cisco Logo are trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and other countries. A listing of Cisco's trademarks can be found at http://www.cisco.com/go/trademarks. Third party trademarks mentioned 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. (1005R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.
Executive Summary

OpenStack is a free and open source Infrastructure-as-a-Service (IaaS) cloud computing project released under the Apache License. It enables enterprises and service providers to offer on-demand computing resources by provisioning and managing large networks of virtual machines. Canonical OpenStack technology on Ubuntu coupled with Ink Tank Ceph storage architecture uses upstream OpenStack open source architecture and enhances it for Enterprise and service provider customers with better support structure. The Cisco Unified Computing System is a next-generation data center platform that unites computing, network, storage access, and virtualization into a single cohesive system. Cisco UCS is an ideal platform for the OpenStack architecture. The combination of Cisco UCS and Canonical OpenStack Platform accelerates your IT Transformation by enabling faster deployments, greater flexibility of choice, efficiency, and lower risk. This Cisco Validate Design document focuses on deploying Canonical OpenStack Platform architecture on Cisco UCS with Ceph storage for enterprise and service provider business segments.

Introduction

OpenStack boasts a massively scalable architecture that can control compute, storage, and networking resources through a unified interface. The OpenStack development community operates on a six-month release cycle with frequent milestones. Their code base is composed of many loosely coupled projects supporting storage, compute, image management, identity, and networking services. OpenStack’s rapid development cycle and architectural complexity create unique challenges for enterprise customers adding OpenStack to their traditional IT portfolios.

Canonical OpenStack technology addresses these challenges. This CVD focuses on Canonical Havana based OpenStack modules running on Ubuntu 12.04 LTS release. Adopters of Canonical OpenStack architecture on Ubuntu enjoy immediate access to bug fixes and critical security patches, tight integration with Ubuntu 12.04 LTS enterprise security features, and a steady release cadence between OpenStack versions. This allows Canonical customers to adopt OpenStack with confidence, at their own pace, and on their own terms.
Virtualization is a key and critical strategic deployment model for reducing the Total Cost of Ownership (TCO) and achieving better utilization of the platform components like hardware, software, network and storage. However choosing the appropriate platform for virtualization can be a tricky task. The platform should be flexible, reliable and cost effective to facilitate the deployment of various enterprise applications onto the virtualization layer. Also, the ability to slice and dice the underlying platform to match the application requirements is essential for a virtualization platform to utilize compute, network and storage resources efficiently. In this regard, implementing OpenStack on Cisco UCS provides a very simplistic yet fully integrated and validated infrastructure for you to deploy VMs in various sizes to suite your application needs.

Target Audience

The reader of this document is expected to have the necessary training and background to install and configure Ubuntu Linux and Cisco Unified Computing System (UCS) and Unified Computing Systems Manager (UCS Manager) as well as a high-level understanding of OpenStack 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 and network and security policies of the customer installation.

Purpose of this Document

This document describes the steps required to deploy and configure Canonical OpenStack architecture on the Cisco UCS platform to a level that will allow for confirmation that the basic components and connections are working correctly. The document addresses small- to medium-sized deployments; however the architecture can be very easily expanded with predictable linear performance. While readers of this document are expected to have sufficient knowledge to install and configure the products used, configuration details that are important to this solution deployment are specifically mentioned.

Solution Overview

Canonical OpenStack Architecture on Cisco UCS Platform

This solution provides an end-to-end architecture with Cisco, Canonical Ubuntu, and OpenStack technologies including Ceph for storage that demonstrate high availability and server redundancy along with ease of deployment and use.

The following are the components used for the design and deployment:

- Cisco Unified Compute System (UCS) 2.2(1)
- Cisco C-series Unified Computing System servers for compute and storage needs
- Cisco UCS VIC adapters
- Cisco Nexus 6000 series switches
- Canonical Ubuntu 12.04 LTS
- OpenStack Havana architecture
Technology Overview

Cisco Unified Computing System

The Cisco Unified Computing System is a next-generation data center platform that unites computing, network, storage access, and virtualization into a single cohesive system.

The main components of the Cisco UCS are:

- **Computing**—The system is based on an entirely new class of computing system that incorporates blade servers based on Intel Xeon E5-2600 V2 Series Processors.
- **Network**—The system is integrated onto a low-latency, lossless, 10-Gbps unified network fabric. This network foundation consolidates LANs, 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**—Cisco C-series servers can host large number of local SATA hard disks. The system provides consolidated access to both SAN storage and Network Attached Storage (NAS) over the unified fabric. By unifying the storage access the Cisco Unified Computing System can access storage over Ethernet, Fibre Channel, Fibre Channel over Ethernet (FCoE), and iSCSI. This provides customers with choice for storage access and investment protection. In addition, the server administrators can pre-assign storage-access policies for system connectivity to storage resources, simplifying storage connectivity, and management for increased productivity.

The Cisco Unified Computing System is designed to deliver:

- A reduced Total Cost of Ownership (TCO) and increased business agility.
- Increased IT staff productivity through just-in-time provisioning and mobility support.
- A cohesive, integrated system which unifies the technology in the data center.
- Industry standards supported by a partner ecosystem of industry leaders.
Cisco Nexus 6001 Switches

The Cisco Nexus 6001 Series Switch is a wire-rate Layer 2 and Layer 3, 48-port 10 Gigabit Ethernet (GE) switch with 40 GE uplinks. It is optimized for high-performance, top-of-rack 10 GE server access and Cisco Fabric Extender (FEX) aggregation. The switch delivers high performance, operational efficiency, and design flexibility for traditional, virtualized, and cloud environments.

Cisco UCS Manager

Cisco Unified Computing System (UCS) Manager provides unified, embedded management of all software and hardware components of the Cisco UCS through an intuitive GUI, a command line interface (CLI), or an XML API. The Cisco UCS Manager provides unified management domain with centralized management capabilities and controls multiple chassis and thousands of virtual machines.

Fabric Interconnect

These devices 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 6248UP Fabric Interconnect

Cisco UCS 6200 Series Fabric Interconnects support the system’s 10-Gbps unified fabric with low-latency, lossless, cut-through switching that supports IP, storage, and management traffic using a single set of cables. The fabric interconnects feature virtual interfaces that terminate both physical and virtual connections equivalently, establishing a virtualization-aware environment in which blade, rack servers, and virtual machines are interconnected using the same mechanisms. The Cisco UCS 6248UP is a 1-RU Fabric Interconnect that features up to 48 universal ports that can support 10 Gigabit Ethernet, Fibre Channel over Ethernet, or native Fibre Channel connectivity. The Cisco UCS 6296UP packs 96 universal ports into only two rack units.
Cisco UCS 2200 Series Fabric Extenders

The Cisco UCS 2200 Series Fabric Extenders multiplex and forward all traffic from blade servers in a chassis to a parent Cisco UCS fabric interconnect over from 10-Gbps unified fabric links. All traffic, even traffic between blades on the same chassis or virtual machines on the same blade, is forwarded to the parent interconnect, where network profiles are managed efficiently and effectively by the fabric interconnect. At the core of the Cisco UCS fabric extender are application-specific integrated circuit (ASIC) processors developed by Cisco that multiplex all traffic.

Fabric Extender

Fabric Extenders are zero-management, low-cost, low-power consuming devices that distribute the system’s connectivity and management planes into rack and blade chassis to scale the system without complexity. Designed never to lose a packet, Cisco fabric extenders eliminate the need for top-of-rack Ethernet and Fibre Channel switches and management modules, dramatically reducing infrastructure cost per server.

Cisco UCS 2232PP Fabric Extender

The Cisco Nexus® 2000 Series Fabric Extenders comprise a category of data center products designed to simplify data center access architecture and operations. The Cisco Nexus 2000 Series uses the Cisco® Fabric Extender architecture to provide a highly scalable unified server-access platform across a range of 100 Megabit Ethernet, Gigabit Ethernet, 10 Gigabit Ethernet, unified fabric, copper and fiber connectivity, rack, and blade server environments. The platform is ideal to support today’s traditional Gigabit Ethernet while allowing transparent migration to 10 Gigabit Ethernet, virtual machine-aware unified fabric technologies.

The Cisco Nexus 2000 Series Fabric Extenders behave as remote line cards for a parent Cisco Nexus switch or Fabric Interconnect. The fabric extenders are essentially extensions of the parent Cisco UCS Fabric Interconnect switch fabric, with the fabric extenders and the parent Cisco Nexus switch together forming a distributed modular system. This architecture enables physical topologies with the flexibility and benefits of both top-of-rack (ToR) and end-of-row (EoR) deployments.

Today’s data centers must have massive scalability to manage the combination of an increasing number of servers and a higher demand for bandwidth from each server. The Cisco Nexus 2000 Series increases the scalability of the access layer to accommodate both sets of demands without increasing management points within the network.
Cisco C220 M3 rack mount servers

Building on the success of the Cisco UCS C220 M2 Rack Servers, the enterprise-class Cisco UCS C220 M3 server further extends the capabilities of the Cisco Unified Computing System portfolio in a 1-rack-unit (1RU) form factor. And with the addition of the Intel® Xeon® processor E5-2600 product family, it delivers significant performance and efficiency gains.

The Cisco UCS C220 M3 also offers up to 256 GB of RAM, eight drives or SSDs, and two 1GE LAN interfaces built into the motherboard, delivering outstanding levels of density and performance in a compact package.

Cisco C240 M3 rack mount servers

The UCS C240 M3 High Density Small Form Factory Disk Drive Model rack server is designed for both performance and expandability over a wide range of storage-intensive infrastructure workloads from big data to collaboration. The enterprise-class UCS C240 M3 server extends the capabilities of Cisco Unified Computing System portfolio in a 2U form factor with the addition of the Intel® Xeon E5-2600 v2 and E5-2600 series processor family CPUs that deliver the best combination of performance, flexibility and efficiency gains. In addition, the UCS C240 M3 server provides 24 DIMM slots, up to 24 drives and 4 x 1 GbE LOM ports to provide outstanding levels of internal memory and storage expandability along with exceptional performance.

Cisco I/O Adapters

The Cisco UCS rack mount server has various Converged Network Adapters (CNA) options. The UCS 1225 Virtual Interface Card (VIC) option is used in this Cisco Validated Design.
A Cisco® innovation, the Cisco UCS Virtual Interface Card (VIC) 1225 (Figure 6) is a dual-port Enhanced Small Form-Factor Pluggable (SFP+) 10 Gigabit Ethernet and Fibre Channel over Ethernet (FCoE)-capable PCI Express (PCIe) card designed exclusively for Cisco UCS C-Series Rack Servers. UCS 1225 VIC provides the capability to create multiple VNICS (up to 128) on the CNA. This allows complete I/O configurations to be provisioned in virtualized or non-virtualized environments using just-in-time provisioning, providing tremendous system flexibility and allowing consolidation of multiple physical adapters.

System security and manageability is improved by providing visibility and portability of network policies and security all the way to the virtual machines. Additional 1225 features like VM-FEX technology and pass-through switching, minimize implementation overhead and complexity.

Figure 6 Cisco UCS 1225 VIC

UCS Single-Wire Management without Fabric Extenders

Starting from Cisco UCS Manager 2.1 supports an additional option to integrate the C-Series Rack-Mount Server with Cisco UCS Manager called “single-wire management”. This option enables Cisco UCS Manager to manage the C-Series Rack-Mount Servers using a single 10 GE link for both management traffic and data traffic. When you use the single-wire management mode, one host facing port on the FEX is sufficient to manage one rack-mount server, instead of the two ports you will use in the Shared-LOM mode. Cisco VIC 1225, Cisco UCS 2232PP FEX and Single-Wire management feature of UCS 2.1 tremendously increases the scale of C-series server manageability. By consuming as little as one port on the UCS Fabric Interconnect, you can manage up to 32 C-series server using single-wire management feature.

While single wire management feature of Cisco UCS Manager 2.1 provides excellent scalability by managing up to 160 servers per UCS domain, the requirement of having a Fabric Extender between C-series serves and the Fabric Interconnects can be viewed as an overhead for small scale deployments. In case of UCS 5548UP Fabric Interconnects, you have 48 ports on fixed module. With maximum 8 x 10GE links between Fabric Interconnect and FEX, you can support up to 32 C-series servers per UCS 2232PP Fabric Extender, providing 4:1 over-subscription. With that configuration, you can scale out to maximum 160 C-series servers with 5 Fabric Extenders per UCS domain, consuming 40 ports on Fabric Interconnect. With maximum 32:1 over-subscription, the number of ports consumed on Fabric Interconnect can be reduced to 5 ports with a single 10GE cable between Fabric Interconnect and FEX.
With UCS 2.2 release, single wire management of C-series servers can be achieved by directly connecting server to Fabric Interconnect, without requiring a Fabric Extender in between. If your UCS pod is not expected to grow beyond 40 C-series servers, then you can opt out the need for Fabric Extender. This would reduce overhead of using Fabric Extenders, saving power, cables and rack space.

**UCS Differentiators**

Cisco 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. Also, a pair of Fabric Interconnects can manage up to 20 chassis, each containing 8 blade servers. This gives enormous scaling on management plane.

2. **Unified fabric**—In UCS, from blade server chassis or rack server fabric-extender to Fabric Interconnect, there is a single Ethernet cable used for LAN, SAN and management traffic. This converged I/O results in reduced cables, SFPs and adapters – reducing capital and operational expenses of overall solution.

3. **Auto Discovery**—By simply inserting the blade server in the chassis or connecting rack server to the fabric extender, discovery and inventory of compute resource occurs automatically without any management intervention. Combination of unified fabric and auto-discovery enables wire-once architecture of UCS, where compute capability of UCS can extending 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 Cisco 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 show cases the policy based resource classification of Cisco UCS Manager.

5. **Combined Rack and Blade server management**—Cisco 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. In this CVD, we are show-casing combination of B and C series servers to demonstrate stateless and form factor independent computing work load.

6. **Model based management architecture**—Cisco UCS Manager architecture and management database is model based and data driven. Open, standard based XML API is provided to operate on the management model. This enables easy and scalable integration of Cisco UCS Manager with other management system, such as VMware vCloud director, Microsoft system center, and Citrix CloudPlatform.

7. **Policies, Pools, Templates**—Management approach in Cisco UCS Manager is based on defining policies, pools and templates, instead of cluttered configuration, which enables simple, loosely coupled, data driven approach in managing compute, network and storage resources.

8. **Loose referential integrity**—In Cisco 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 flexibilities 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 Cisco 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 other policy by name is resolved in the organization hierarchy with closest policy match. If no policy with specific name is found in the hierarchy till root organization, then special policy named “default” is searched. This policy resolution practice enables automation friendly management APIs and provides great flexibilities to owners of different organizations.

10. Service profiles and stateless computing—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—Combination of policies, pools and templates, loose referential integrity, policy resolution in organization hierarchy and service profile based approach to compute resources makes Cisco UCS Manager inherently friendly to multi-tenant environment typically observed in private and public clouds.

12. Virtualization aware network—VM-FEX technology makes access layer of network 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 offloads 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.

13. Simplified QoS—Even though fibre-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 Cisco UCS Manager by representing all system classes in one GUI panel.

**Canonical Ubuntu OpenStack Architecture**

Canonical OpenStack Platform on Canonical Ubuntu 12.04 provides the foundation to build private or public Infrastructure-as-a-Service (IaaS) for cloud-enabled workloads. It allows organizations to leverage OpenStack, the largest and fastest growing open source cloud infrastructure project, while maintaining the security, stability, and enterprise readiness of a platform built on Canonical Ubuntu 12.04.

Canonical Ubuntu OpenStack Platform gives organizations a truly open framework for hosting cloud workloads, delivered by Canonical support system for maximum flexibility and cost effectiveness. In conjunction with other Ubuntu technologies, Canonical Ubuntu OpenStack Platform allows organizations to move from traditional workloads to cloud-enabled workloads on their own terms and timeline, as their applications require. Canonical frees organizations from proprietary lock-in, and allows them to move to open technologies while maintaining their existing infrastructure investments.

Unlike other OpenStack distributions, Canonical Ubuntu OpenStack Platform provides a certified ecosystem of hardware, software, and services, an enterprise lifecycle that extends the community OpenStack release cycle, and Canonical support on both the OpenStack modules and their underlying Linux dependencies. Canonical delivers long-term commitment and value from a proven enterprise software partner so organizations can take advantage of the fast pace of OpenStack development without risking the stability and supportability of their production environments.
Canonical Ubuntu OpenStack Havana Software Components

This CVD focuses on Canonical OpenStack software components based on the upstream “Havana” OpenStack release. Ubuntu is the de facto Linux distribution to deploy OpenStack among Service Providers and Enterprise customers. Following few subsections cover key software components involved in OpenStack.

**Identity Service (Keystone)**

This is a central authentication and authorization mechanism for all OpenStack users and services. It supports multiple forms of authentication including standard user name and password credentials, token-based systems and AWS-style logins that use public/private key pairs. It can also integrate with existing directory services such as LDAP.

The Identity service catalog lists all of the services deployed in an OpenStack cloud and manages authentication for them through endpoints. An endpoint is a network address where a service listens for requests. The Identity service provides each OpenStack service – such as Image, Compute, or Block Storage -- with one or more endpoints.

The Identity service uses tenants to group or isolate resources. By default users in one tenant can’t access resources in another even if they reside within the same OpenStack cloud deployment or physical host. The Identity service issues tokens to authenticated users. The endpoints validate the token before allowing user access. User accounts are associated with roles that define their access credentials. Multiple users can share the same role within a tenant.

The Identity Service is comprised of the keystone service, which responds to service requests, places messages in queue, grants access tokens, and updates the state database.
Image Service (Glance)

This service discovers, registers, and delivers virtual machine images. They can be copied via snapshot and immediately stored as the basis for new instance deployments. Stored images allow OpenStack users and administrators to provision multiple servers quickly and consistently. The Image Service API provides a standard RESTful interface for querying information about the images.

By default the Image Service stores images in the /var/lib/glance/images directory of the local server’s filesystem where Glance is installed. The Glance API can also be configured to cache images in order to reduce image staging time. The Image Service supports multiple back end storage technologies including Swift (the OpenStack Object Storage service), and Amazon S3.

The Image service is composed of the openstack-glance-api that delivers image information from the registry service, and the openstack-glance-registry which manages the metadata associated with each image.

Compute Service (Nova)

OpenStack Compute provisions and manages large networks of virtual machines. It is the backbone of OpenStack IaaS functionality. OpenStack Compute scales horizontally on standard hardware enabling the favorable economics of cloud computing. Users and administrators interact with the compute fabric via a web interface and command line tools.

Key features of OpenStack Compute include:

- Distributed and asynchronous architecture, allowing scale out fault tolerance for virtual machine instance management
- Management of commoditized virtual server resources, where predefined virtual hardware profiles for guests can be assigned to new instances at launch
- Tenants to separate and control access to compute resources
- VNC access to instances via web browsers

OpenStack Compute is composed of many services that work together to provide the full functionality. The openstack-nova-cert and openstack-nova-consoleauth services handle authorization. The openstack-nova-api responds to service requests and the openstack-nova-scheduler dispatches the requests to the message queue. The openstack-nova-conductor service updates the state database which limits direct access to the state database by compute nodes for increased security. The openstack-nova-compute service creates and terminates virtual machine instances on the compute nodes. Finally, openstack-nova-novncproxy provides a VNC proxy for console access to virtual machines via a standard web browser.

Block Storage (Cinder)

While the OpenStack Compute service provisions ephemeral storage for deployed instances based on their hardware profiles, the OpenStack Block Storage service provides compute instances with persistent block storage. Block storage is appropriate for performance sensitive scenarios such as databases or frequently accessed file systems. Persistent block storage can survive instance termination. It can also be moved between instances like any external storage device. This service can be backed by a variety of enterprise storage platforms or simple NFS servers. This service’s features include:

- Persistent block storage devices for compute instances
- Self-service user creation, attachment, and deletion
- A unified interface for numerous storage platforms
- Volume snapshots

The Block Storage service is comprised of openstack-cinder-api which responds to service requests and openstack-cinder-scheduler which assigns tasks to the queue. The openstack-cinder-volume service interacts with various storage providers to allocate block storage for virtual machines. By default the Block Storage server shares local storage via the ISCSI tgt daemon.

**Network Service (Neutron)**

OpenStack Networking is a scalable API-driven service for managing networks and IP addresses. OpenStack Networking gives users self-service control over their network configurations. Users can define, separate, and join networks on demand. This allows for flexible network models that can be adapted to fit the requirements of different applications.

OpenStack Networking has a pluggable architecture that supports numerous physical networking technologies as well as native Linux networking mechanisms including openvswitch and linuxbridge. OpenStack Networking is composed of several services. The quantum-server exposes the API and responds to user requests. The quantum-l3-agent provides L3 functionality, such as routing, through interaction with the other networking plug-ins and agents. The quantum-dhcp-agent provides DHCP to tenant networks. There are also a series of network agents that perform local networking configuration for the node’s virtual machines.

---

**Note**

In previous OpenStack versions the Network Service was named Quantum. From the Grizzly release Quantum was renamed to Neutron. However, many of the command line utilities retain the legacy name.

**Dashboard (Horizon)**

The OpenStack Dashboard is an extensible web-based application that allows cloud administrators and users to control and provision compute, storage, and networking resources. Administrators can use the Dashboard to view the state of the cloud, create users, assign them to tenants, and set resource limits.

The OpenStack Dashboard runs as an Apache HTTP server via the httpd service.

---

**Note**

Both the Dashboard and command line tools can be used to manage an OpenStack environment. This document focuses on the command line tools because they offer more granular control and insight into OpenStack functionality.

**Ceph Storage for OpenStack**

Ceph is a massively scalable, open source, software defined storage system. It offers an object store and a network block device, unified for the cloud. The platform is capable of auto-scaling to the exabyte level and beyond, it runs on commodity hardware, is self-healing and self-managing, and has no single point of failure. Ceph is in the Linux kernel, and has been integrated with OpenStack since the Folsom release. Ceph is ideal for creating flexible, easy to operate object and block cloud storage.
Unlike every other storage solution for OpenStack, Ceph uniquely combines object and block into one complete storage powerhouse for all your OpenStack needs. Ceph is a total replacement for Swift with distinctive features such as intelligent nodes and a revolutionary deterministic placement algorithm, along with a fully integrated network block device for Cinder. Ceph fully distributed storage cluster and block device decouple compute from storage in OpenStack, allowing mobility of virtual machines across your entire cluster. Ceph block device also provides copy on write cloning that enables you to quickly create a thousand VMs from a single master image, requiring only enough space to store their subsequent changes.

**Canonical Ubuntu 12.04 LTS Release**

Ubuntu 12.04 also known by its code name “Precise Pangolin”, is a Long Term Support (LTS) release from Canonical. The support for Ubuntu 12.04 is expected to continue till April 2017, hence providing a long term robust support framework for customers. For open source projects, support is a crucial component, and Canonical provides enterprise level scale, stability and support for underlying Operating System as well as OpenStack components for cloud deployment on Ubuntu.

The Ubuntu team broke new ground in committing to a program of scheduled releases on a predictable six month basis. It was decided that every fourth release, issued on a two-year basis, would receive long-term support (LTS). LTS releases are typically used for large-scale deployments.

Ubuntu is different from the commercial Linux offerings that preceded it because it doesn't divide its efforts between a high-quality commercial version and a free 'community' version. The commercial and community teams collaborate to produce a single, high-quality release, which receives ongoing maintenance for a defined period. Both the release and ongoing updates are freely available to all users.
Cisco Build Node

This CVD demonstrates use of Cisco Build Node for ease of deployment of OpenStack components. Build Node integrates with Cisco UCS Manager in a smooth fashion, so administrator can deploy the whole OpenStack cloud from one place without managing multiple components individually. Build Node acts as a proxy to download all OpenStack components at one location, holds the ISO image of Ubuntu servers and deploys the images on UCS servers using Cobbler and Puppet. Build Node scripts use north bound XML API of Cisco UCS Manager and auto configures the service profile templates, service profiles and other policies on Cisco UCS Manager. After configuring the Cisco UCS Manager, the scripts listens for key server discovery and association events from Cisco UCS Manager and trigger installation of operating system on the servers. The servers are also automatically classified into various roles, such as storage node, controller node or compute node, and right set of OpenStack software is installed on the node depending on its role using cobbler and puppet. All of these automation makes hassle-free deployment of OpenStack on UCS servers.

Architectural overview

This CVD focuses on the architecture for Canonical Ubuntu OpenStack on UCS platform using Cisco UCS C-series servers for storage. Cisco UCS C220 M3 servers are used as compute and controller nodes and UCS C240 M3 servers are used as storage nodes. Storage high availability and redundancy are achieved using Ceph storage services on OpenStack. UCS C-series servers are managed by Cisco UCS Manager, which provides ease of infrastructure management and built-in network high availability.

Table 1 lists the various hardware and software components which occupy different tiers of the architecture under test.

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Name</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cisco</td>
<td>Nexus 6001 switch</td>
<td>6.0(2)N1(2)</td>
<td>Nexus 6001 switch</td>
</tr>
<tr>
<td>Cisco</td>
<td>Cisco UCS Manager</td>
<td>2.2(1b)</td>
<td>UCS Manager</td>
</tr>
<tr>
<td>Cisco</td>
<td>UCS 6248UP Fabric Interconnect</td>
<td>5.2(3)N2(2.21b)</td>
<td>UCS Fabric Interconnects</td>
</tr>
<tr>
<td>Cisco</td>
<td>UCS 2232PP FEX</td>
<td>5.2(3)N2(2.21b)</td>
<td>UCS Fabric Extenders</td>
</tr>
<tr>
<td>Cisco</td>
<td>UCS C220 M3 servers</td>
<td>1.5(4) – CIMC C220M3.1.5.4f.0.111320130449 – BIOS</td>
<td>Cisco UCS C220 M3 rack servers</td>
</tr>
<tr>
<td>Cisco</td>
<td>UCS C240 M3 servers</td>
<td>1.5(4) – CIMC C240M3.1.5.4f.0.111320130505 – BIOS</td>
<td>Cisco UCS C240 M3 rack servers</td>
</tr>
<tr>
<td>Cisco</td>
<td>UCS VIC 1225</td>
<td>2.2(1b)</td>
<td>Cisco UCS VIC adapter</td>
</tr>
<tr>
<td>Canonical</td>
<td>Ubuntu Operating System</td>
<td>12.04.4 LTS</td>
<td>Canonical Ubuntu Linux Operating System</td>
</tr>
</tbody>
</table>
Table 2 outlines the C220 M3 server configuration, used as compute nodes of the architecture. The table shows the configuration on a per server basis.

**Table 2  C220 M3 server configuration**

<table>
<thead>
<tr>
<th>Component</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (RAM)</td>
<td>128 GB (16 x 8MB DIMM)</td>
</tr>
<tr>
<td>Processor</td>
<td>2 x Intel(R) Xeon(R) E5-2660 v2 CPUs, 2.2 GHz, 10 cores, 20 threads</td>
</tr>
<tr>
<td>Local storage</td>
<td>Cisco UCS RAID SAS 2008M-8i Mezzanine Card, with 4, 300GB disks for RAID 5 configuration each.</td>
</tr>
</tbody>
</table>

Table 3 outlines the C240 M3 server configuration, used as the storage nodes of the architecture. The table shows the configuration on a per server basis.

**Table 3  C240 M3 server configuration**

<table>
<thead>
<tr>
<th>Component</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (RAM)</td>
<td>128 GB (16 x 8MB DIMM)</td>
</tr>
<tr>
<td>Processor</td>
<td>2 x Intel(R) Xeon(R) E5-2660 v2 CPUs, 2.2 GHz, 10 cores, 20 threads</td>
</tr>
<tr>
<td>Local storage</td>
<td>Cisco UCS LSI 6G MegaRAID SAS 9266-8i controller card, with 24, 1TB 7.2K RPM SATA disks for RAID0 individual disk each</td>
</tr>
</tbody>
</table>
Following are the high-level design points of Ubuntu OpenStack architecture on UCS Platform (Figure 9):

- Redundant UCS Fabric Interconnects, Fabric Extenders and multiple cables provide network high availability.
- Multiple hard disks per storage node combined with multiple storage nodes provide storage high availability through OpenStack Ceph module.
- Build Node has L2 connectivity to all compute as well as storage nodes of UCS C-series servers, as well as L2/L3 accessibility to Cisco UCS Manager OOB management IP interface.
- Infrastructure network is on a separate 1GE network. Out of band UCS management and other legacy infrastructure components, such as Syslog server, are connected to infrastructure network.
- This design does not dictate or require any specific layout of infrastructure network. The Out Of Band Cisco UCS Manager access, hosting of supporting infrastructure such as Syslog server are hosted on infrastructure network. However, design does require accessibility of certain VLANs from the infrastructure network to reach the servers.

**OpenStack services placement**

Table 4 shows the final service placement for all OpenStack services. This CVD show cases the “compressed-ha” scenario of Cisco OpenStack deployment scripts. There are three controller nodes, together they provide highly available OpenStack control plane. Controller nodes host all control plane
services such as Horizon Dashboard backend, Identity service, all API services, and Ceph Monitor cluster. Given that the controller nodes have access to CPU, memory, and network resources, they can also host VMs and therefore play the role of compute nodes in addition to their controller nodes function.

Compute and Storage nodes are quite self-explanatory, compute nodes host OpenStack VM instances while storage nodes manage the storage disks for the VMs using Ceph OSD clusters. Both compute and storage nodes can be added as the need of the cloud resources grows, keeping controller nodes count at 3.

Build Node plays a crucial role in deploying the Ubuntu OpenStack on UCS domain by hosting various scripts, puppet, cobbler and UCS Python SDK and integration scripts and acting as OpenStack repository proxy.

**Virtual Networking**

This architecture demonstrates use and benefits of Adapter-FEX technology using Cisco UCS VIC adapter. Each C220 M3 and C240 M3 server has one Cisco VIC 1225 physical adapter with two 10 GE links going to fabric A and fabric B for high availability. Cisco UCS VIC 1225 can present multiple virtual Network Interface Cards (vNICs) to the hypervisor with multiple virtual interfaces (one on each fabric) in active/passive mode. These vNICs are capable to do fabric failover, so if the Fabric Extender of Fabric Interconnect reboots or all the uplinks on the Fabric Interconnect are lost, the vNIC would move traffic from fabric A to fabric B (or vice-a-versa) transparently. The MAC addresses to these vNICs are assigned using MAC address pool defined on the Cisco UCS Manager.

- All the OpenStack nodes would have at least following two vNICs:
- eth0 – public or management network: This is the interface through which you can access Ubuntu servers from outside of the pod. The same network is used by various OpenStack components to communicate among various OpenStack services components. Build Node’s eth0 interface and all the eth0 interfaces of OpenStack nodes must be in the same VLANs for Pxe booting of the servers.

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucs-openstack-buildnode</td>
<td>Build Node</td>
<td>Hosts Cisco build node scripts, puppet and cobbler based UCS OpenStack cloud deployment</td>
</tr>
<tr>
<td>controller-node1</td>
<td>Controller, Compute</td>
<td>Runs all openstack controller services, Ceph monitor services in HA mode with other controller nodes. Also runs compute services to deploy VMs</td>
</tr>
<tr>
<td>controller-node2</td>
<td>Controller, Compute</td>
<td>Runs all openstack controller services, Ceph monitor services in HA mode with other controller nodes. Also runs compute services to deploy VMs</td>
</tr>
<tr>
<td>controller-node3</td>
<td>Controller, Compute</td>
<td>Runs all openstack controller services, Ceph monitor services in HA mode with other controller nodes. Also runs compute services to deploy VMs</td>
</tr>
<tr>
<td>compute-node1</td>
<td>Compute</td>
<td>Compute node for OpenStack VMs.</td>
</tr>
<tr>
<td>compute-node 2</td>
<td>Compute</td>
<td>Compute node for OpenStack VMs.</td>
</tr>
<tr>
<td>compute-node 3</td>
<td>Compute</td>
<td>Compute node for OpenStack VMs.</td>
</tr>
<tr>
<td>storage-node1</td>
<td>Storage</td>
<td>Ceph OSD Storage Node</td>
</tr>
<tr>
<td>storage-node2</td>
<td>Storage</td>
<td>Ceph OSD Storage Node</td>
</tr>
<tr>
<td>storage-node3</td>
<td>Storage</td>
<td>Ceph OSD Storage Node</td>
</tr>
</tbody>
</table>

**Table 4 Service Placement**

---

Ubuntu OpenStack Architecture on Cisco UCS Platform
• eth1 – private or external network: This interface would be taken over by the virtual networking component of Nova. You cannot use this interface to login to Ubuntu server. This interface would carry multiple VLANs used by various tenants within OpenStack network. If tenants’ VMs require access to external network, which in most cases many of the VMs would do, the eth1 interface must also carry a VLAN to access external network / Internet. VM to VM traffic (east/west traffic) can be completely contained within OpenStack pod, while VM to Internet traffic (north/south traffic) would be routed and NATed through software router within OpenStack and go through external VLAN on this interface.

• In addition to above mentioned two vNICs, Storage nodes would have one more vNIC:

• eth2 – Storage Replication network: This interface would be used by Ceph OSD processes to replicate user data across multiple storage nodes.

• In the hypervisor layer, this architecture is using Neutron (Quantum) networking layer, with Open-vSwitch for virtual networking. Different VLANs are used for different tenants for logical separation of domains. Within a given tenant’s realm, different VLANs can be used on per tier basis too in case of multi-tier applications. In other words, architecture does not dictate one VLAN per tenant.

Storage Virtualization

Storage Layout

There are 24 one TB SATA disks per C240 M3 server. Default configuration demonstrated in this CVD puts each disk into its own RAID0 configuration, out of which, first disk is selected as the bootable device. Ubuntu 12.04 image is installed on the bootable disk through PXE booting. All remaining 23 disks are available for Ceph. In Linux terminology, /dev/sda is where OS is installed and the disks /dev/sdb to /dev/sdx are available to Ceph as storage devices.

The Ceph Storage Cluster is the foundation for all Ceph deployments. Based upon RADOS, Ceph Storage Clusters consist of two types of daemons: a Ceph OSD Daemon (OSD) stores data as objects on a storage node; and a Ceph Monitor maintains a master copy of the cluster map. A Ceph Storage Cluster may contain thousands of storage nodes. In our architecture, C240 M3 servers act as Ceph OSD nodes, while 3 controller nodes run as Ceph monitor nodes.

The Ceph Filesystem, Ceph Object Storage and Ceph Block Devices read data from and write data to the Ceph Storage Cluster. The Ceph Filesystem (Ceph FS) is a POSIX-compliant file system that uses a Ceph Storage Cluster to store its data. The Ceph file system uses the same Ceph Storage Cluster system as Ceph Block Devices, Ceph Object Storage with its S3 and Swift APIs, or native bindings (librados). Block-based storage interfaces are the most common way to store data with rotating media such as hard disks, CDs, floppy disks, and even traditional 9-track tape. The ubiquity of block device interfaces makes a virtual block device an ideal candidate to interact with a mass data storage system like Ceph.

Ceph block devices are thin-provisioned, resizable and store data striped over multiple OSDs in a Ceph cluster. Ceph block devices leverage RADOS capabilities such as snapshotting, replication and consistency. Ceph RADOS Block Devices (RBD) interact with OSDs using kernel modules or the librbd library. Ceph block devices deliver high performance with infinite scalability to kernel modules, or to KVMs such as Qemu, and cloud-based computing systems like OpenStack and CloudStack that rely on libvirt and Qemu to integrate with Ceph block devices. You can use the same cluster to operate the Ceph RADOS Gateway, the Ceph FS file system, and Ceph block devices simultaneously.
Service Profile Design

This architecture implements following design steps to truly achieve stateless computing on the servers:

- Service profiles are derived from service profile templates for consistency.
- The Ubuntu host uses following identities in this architecture:
  - Host UUID
  - Mac Addresses: one per each vNIC on the server

These identifiers are defined in their respective identifier pools and the pool names are referred in the service profile template.

- Server pools are defined with automatic qualification policy and criteria. Servers with large amount of local hard disks are automatically put in the storage nodes pool and servers with large number of cores are put in the compute nodes pool as and when they are fully discovered by Cisco UCS Manager. This eliminates the need to manually assign servers to server pool.
- Service profile templates are associated to the server pool. This eliminates the need to individually associating service profiles to physical servers.

Given this design and capabilities of UCS and Cisco UCS Manager, a new server can be procured within minutes if the scale needs to be increased or if a server needs to be replaced by different hardware. This is very critical for the cloud deployments, where cloud must be able to grow or shrink in capacity as per the needs. In case, if a rack server has physical fault (faulty memory, or PSU or fan, for example), using following steps then a new server can be procured within minutes:

- Move VMs running on faulty server to other healthy servers on the cluster using OpenStack commands, if the faulty server is playing the role of compute node.
- Decommission the faulty server using Cisco UCS Manager. This would disassociate the service profile from the faulty server. Physically remove the server for replacement of faulty hardware (or to completely remove the faulty server).
- Physically install the new server and connect it to the Fabric Extenders (or Fabric Interconnect, as per your design). Let the new server be discovered by Cisco UCS Manager and automatically reclassified in to right server pool. (If you replaced a faulty component of the server and not the entire server, then you would need to re-commission the server from Cisco UCS Manager manually.)
- Service profile instance is associated to server pool, so newly discovered server would be automatically associated.
- The new server would assume the role of the old server with all the identifiers intact.
- UCS integration scripts listening for service profile association event on the Build Node would reinstall Ubuntu on the new server and install necessary OpenStack components on it.

Coupled with Cisco UCS Manager features and Build Node automation, this architecture achieves the true statelessness of the computing in the data-center. If there are enough identifiers in all the id-pools, and if more servers are attached to UCS system in future, more service profiles can be derived from the service profile template and the private cloud infrastructure can be easily expanded.

Network High Availability Design

Following are the key aspects of this solution:

- Cisco adapter-FEX technology to introduce virtual NICs to host OS
- Fabric failover feature of adapter-FEX is exploited to provide high availability
• Two 10GE links between Fabric Interconnect and FEX provides enough bandwidth over-subscription for the
• given size of cloud. The over-subscription can be reduced by adding more 10GE links between Fabric Interconnect and FEX if needed by the VMs running on the hosts.
• Two vNICs per host – one for private network within the OpenStack environment and one for the public access of the Linux hosts. Additional vNIC for storage replication for the storage nodes.
• Control, management and storage replication traffic to go on fabric A while tenant’s traffic to go over fabric B. Fabric failover feature would move the traffic from one fabric to another, if the primary fabric fails.
• All OpenStack services are running on more than one controller node to make them highly available.

Storage High Availability Design

Ceph OSD nodes (storage nodes) provide storage high availability for all type of cloud storage – block storage, object storage as well as image repository storage. The data is replicated over multiple disks on a given node as well as multiple nodes – so, the cloud provides both disk level and node level high-availability. As Ceph provides backend for Cinder, Swift and Glance, and takes care of data replication, there is no need for RAID configuration across multiple disks as explained in Storage Virtualization section.

For ease of deployment, all the disks on C240 M3 servers are put into individual RAID0 mode. This means that even the bootable disk where Ubuntu is installed is also in RAID0 configuration. Should the bootable disk fail on a Storage Node, the whole node would become unusable. The cloud is still highly available, as Ceph maintains multiple copies of data across the nodes. In order to restore the faulty Storage Node, you can leverage the stateless computing feature of UCS domain and replace the node as explained in the “Service Profile Design” section.

However, if you want to choose manual configuration of local disks for additional HA, then you can put first two disks of C240 M3 server in a RAID1 configuration, where you can install Ubuntu and keep remaining 22 disks in individual RAID0 configuration where Ceph would store data. “Appendix” section on page 59 shows how to manually configure local disks using WebBIOS GUI for this alternative storage high availability for operating system.

Next sub section goes into sizing guidelines of the Cisco solution for architectures outlined here.

Sizing Guideline

In any discussion about virtual infrastructures, it is important to first define a reference workload. Not all servers perform the same tasks, and it is impractical to build a reference that takes into account every possible combination of workload characteristics.

Defining the Reference Workload

To simplify the discussion, we have defined a representative customer reference workload. By comparing your actual customer usage to this reference workload, you can extrapolate which reference architecture to choose.

OpenStack defines various reference VMs as shown in Table 5.
This specification for a virtual machine is not intended to represent any specific application. Rather, it represents a single common point of reference to measure other virtual machines.

You must design your cloud to provide N + 1 hosts high availability. In order to do so, consider the largest resource required by all the VMs, divide it by the single physical server resources and round it up. This would give you required number of hosts. Add one more host to provide N+1 HA.

For example, all the instances required to run on your cloud would require combined 620 GB of RAM. With 128 GB RAM per server, this would require 5 servers. To provide N + 1 HA, you would need 6 compute nodes and divide the load across all the hosts. In this case, if one of the hosts has to go down for maintenance, remaining servers can still carry the load of all instances. This example assumes that RAM requirements is the highest across all instances.

---

**Configuration Guidelines**

The configuration for Ubuntu OpenStack architecture on UCS Platform is divided in to following steps:

1. Connecting Network Cables, page 26
2. Cisco Nexus 6001 Switches, page 9
3. Initial Configuration of Cisco UCS Manager, page 28
4. Install Build Node, page 30
5. Validating Ubuntu OpenStack on UCS Platform, page 50

---

**Connecting Network Cables**

See the Cisco UCS Fabric Interconnect, Fabric Extenders, and C- Series Server configuration guide for detailed information on mounting the hardware on the rack. Figure 10 shows the connectivity details for the architecture covered in this document.

There are four major cabling sections in this architecture listed below (see Figure 10). The network connectivity is quite simple and self-explanatory. Following are brief explanation for each:

1. Fabric Interconnects Upstream Connectivity (shown in purple)—From each Fabric Interconnect, at least one 10 GE link is connected to each Cisco Nexus 6001 switch. Note that there is Fabric Interconnect to Fabric Interconnect peer 10 GE links. However, there should be at least two 10 GE links between two Nexus 6001 switches for high availability. Connectivity shown here is the minimum number of links required, you may want to add additional links if your cloud demands more network traffic.
2. Fabric Interconnects to Fabric Extenders links (shown in blue)—Each Fabric Interconnect claims one Fabric Extender, and minimum two links are recommended between Fabric Interconnect and FEX. Note that a given FEX connects to only one Fabric Interconnect, there is “cross-connections” between Fabric Interconnect and FEX, as FEX behaves like a line card to the Fabric Interconnect, and a line card can be virtually inserted in only one Fabric Interconnect.

3. Fabric Extenders to C-series server links (shown in green)—Each C-series server (both C220 M3 and C240 M3) connects to each Fabric Extender using a 10 GE link. On server side, you connect the link to the Cisco VIC adapter (as opposed to LOM ports).

4. Infrastructure connectivity (not shown)—In addition to the connections listed above, two Fabric Interconnects are connected to each other through a pair of 100Mbps umbilical cord for management plane sync. Each Fabric Interconnects are connected to infrastructure network for Out-Of-Bound management connectivity (mgmt0 interfaces of the Fabric Interconnects). Additional C220 M3 server, which serves as a Build Node is also accessible to this pod. Typically, Build Node server is connected to Cisco Nexus 6000 switches, however, it is not a requirement.

Note: Build Node should not be connected to the Cisco UCS Fabric Interconnects or Cisco UCS Fabric Extenders.

**Figure 10  Detailed Connectivity Diagram of the Architecture**

Connect all the cables as outlined above, and you would be ready to configure the Nexus 6001 switches and Cisco UCS Manager.
Configure Nexus 6001 Switches

First step of configuring the OpenStack pod is to configure Nexus 6001 switches. Following steps walks you through the necessary configuration of the switches.

1. VLAN configuration: Configure public, private and storage VLANs on each switch. You may need to add additional tenant VLANs later as needed. Following CLI shows how to configure a single VLAN, repeat these steps for each VLAN in the setup. Refer to Table 10 for the VLANs used in the architecture.

   N6K-A# configure terminal
   Enter configuration commands, one per line. End with CNTL/Z.
   N6K-A(config)# vlan 60
   N6K-A(config-vlan)# name private-network
   N6K-A(config-vlan)# end
   N6K-A#

2. For each port connected to Fabric Interconnect, change the mode to trunk, make management VLAN as native VLAN and prune unnecessary VLANs by restricting allowed VLAN list to only required VLANs as shown below. This configuration should be repeated for each port connected to Fabric Interconnect on each switch.

   N6K-A(config)# interface ethernet 1/15
   N6K-A(config-if)# switchport mode trunk
   N6K-A(config-if)# switchport trunk native vlan 602
   N6K-A(config-if)# switchport trunk allowed vlan 602, 60-65
   N6K-A(config-if)# end
   N6K-A#

3. Configure port-channel for links between two Nexus 6001 switches. This configuration should be repeated on each switch.

   N6K-A(config)# interface port-channel 1
   N6K-A(config-if)# exit
   N6K-A(config)# interface ethernet 1/19-20
   N6K-A(config-if-range)# channel-group 1 mode active
   N6K-A(config-if-range)# exit
   N6K-A(config)# interface port-channel 1
   N6K-A(config-if)# switchport mode trunk
   N6K-A(config-if)# switchport trunk native vlan 602
   N6K-A(config-if)# switchport trunk allowed vlan 602, 60-65
   N6K-A(config-if)# exit
   N6K-A(config)#

4. At the end of all the configuration, make sure to save the config on each switch as shown below.

   N6K-A# copy running-config startup-config
   [########################################] 100%
   Copy complete, now saving to disk (please wait)...
   N6K-A#

That is a minimum required configuration for Nexus 6001 switches. You can also configure virtual port-channels (or vpc for short) for the links connected from Fabric Interconnects to Nexus 6001 switches for improved link level redundancy, however given the fact that UCS Fabric Interconnects run in end-host mode, vpc doesn’t add a lot of value add and upstream port-channel configuration is not automated in the Build Node script as of now. Given that, for simplicity reasons, this architecture is not choosing vpc configuration.

Initial Configuration of Cisco UCS Manager

At this point of time, the Cisco UCS Fabric Interconnect, Fabric Extender, and C-Series server must be connected appropriately as per “Connecting Network Cables” section on page 26. Two 100 Mbps Ethernet cables must be connected between two Fabric Interconnects for management pairing. Two
redundant power supplies are provided per Fabric Interconnect, it is highly recommended that both are plugged in, ideally drawing power from two different power strips. Connect mgmt0 interfaces of each Fabric Interconnect to the infrastructure network, and put the switch port connected to Fabric Interconnect in access mode with access VLAN as management VLAN. Now follow these steps to perform initial configuration of Fabric Interconnects:

1. Attach RJ-45 serial console cable to the first Fabric Interconnect, and connect the other end to the serial port of laptop. Configure password for the admin account, fabric ID “A”, UCS system name, management IP address, subnet mask and default gateway and cluster IP address (or Cisco UCS Manager Virtual IP address), as the initial configuration script walks you through the configuration as shown in Figure 11. Save the configuration, which would eventually lead to Cisco UCS Manager CLI login prompt.

![Figure 11 Connect RJ-45 Serial Console Cable](image)

2. Disconnect the RJ-45 serial console from the Fabric Interconnect that you just configured and attach it to the other Fabric Interconnect. Other Fabric Interconnect would detect that its peer has been configured, and would prompt you to just join the cluster. Only information you need to provide is the Fabric Interconnect specific management IP address, subnet mask and default gateway, as shown in Figure 12. Save the configuration.
Once initial configurations on both Fabric Interconnects are completed, you can disconnect the serial console cable. Now, Cisco UCS Manager would be accessible through web interface or SSH. Connect to Cisco UCS Manager using SSH, and see the HA status. Since there is a rack server or a blade server chassis connected between the two Fabric Interconnects, the status is shown as “HA NOT READY”. You need to run the `show cluster state` command to confirm that both the Fabric Interconnects A and B are in the UP state and the status shows “HA READY” as shown in Figure 13.

### Install Build Node

1. On the Build Node server, run the following commands to install Ubuntu using defaults. Ensure that hostname does not contain any capital letters.

   ```bash
   openstack-admin@ucs-openstack-buildnode:~$ uname -a
   Linux ucs-openstack-buildnode 3.11.0-15-generic #25~precise1-Ubuntu SMP Thu Jan 30 17:39:31 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
   openstack-admin@ucs-openstack-buildnode:~$ lsb_release -a
   No LSB modules are available. 
   Distributor ID: Ubuntu  
   Description: Ubuntu 12.04.4 LTS  
   Release: 12.04  
   Codename: precise  
   openstack-admin@ucs-openstack-buildnode:~$
   
   The installer would configure basic networking (including IP address, subnet mask, default gateway, DNS server and Proxy Server). It is highly recommended to use an NTP server common across all devices in the cloud, including the build node. Run the following commands to install and configure NTP on Ubuntu.

   ```bash
   # apt-get install ntp
   Reading package lists... Done
   Building dependency tree
   ```
Reading state information... Done
The following extra packages will be installed:
  libopts25
Suggested packages:
  ntp-doc
The following NEW packages will be installed:
  libopts25 ntp
0 upgraded, 2 newly installed, 0 to remove and 89 not upgraded.
Need to get 672 kB of archives.
After this operation, 1,708 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise/main libopts25 amd64
  1:5.12-0.1ubuntu1 [59.9 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main ntp amd64
  1:4.2.6.p3+dfsg-1ubuntu3.1 [612 kB]
Fetched 672 kB in 2s (227 kB/s)
Selecting previously unselected package libopts25.
(Reading database ... 79931 files and directories currently installed.)
Unpacking libopts25 (from .../libopts25_1%3a5.12-0.1ubuntu1_amd64.deb) ...
Selecting previously unselected package ntp.
Unpacking ntp (from .../ntp_1%3a4.2.6.p3+dfsg-1ubuntu3.1_amd64.deb) ...
Processing triggers for ureadahead ...
Processing triggers for man-db ...
Setting up libopts25 (1:5.12-0.1ubuntu1) ...
Setting up ntp (1:4.2.6.p3+dfsg-1ubuntu3.1) ...
* Starting NTP server ntpd [ OK ]
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place

Note
To add your own local NTP server, edit the /etc/ntp.conf file and add the details under server
<ntp-server-hostname> section.

2. Run the following commands to download and install git. You can export http_proxy and/or
https_proxy environment variables if you are behind a proxy.

    openstack-admin@ucs-openstack-buildnode:~$ sudo su -
    [sudo] password for openstack-admin:
    root@ucs-openstack-buildnode:~# export PS1="#
    # apt-get install -y git
    Reading package lists... Done
    Building dependency tree
    Reading state information... Done
    The following extra packages will be installed:
    git-man liberror-perl
Suggested packages:
  git-daemon-run git-daemon-sysvinit git-doc git-el git-arch git-cvs git-svn
  git-email git-gui gtk gitweb
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 71 not upgraded.
Need to get 6,741 kB of archives.
After this operation, 15.2 MB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise/main liberror-perl all 0.17-1
  [23.8 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ precise/main git-man all 1:1.7.9.5-1
  [630 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu/ precise/main git amd64 1:1.7.9.5-1
  [6,087 kB]
Fetched 6,741 kB in 45s (149 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 50970 files and directories currently installed.)
Unpacking liberror-perl (from .../liberror-perl_0.17-1_all.deb) ...
Selecting previously unselected package git-man.
Unpacking git-man (from .../git-man_1%3a1.7.9.5-1_all.deb) ...
Selecting previously unselected package git.
Unpacking git (from .../git_1%3a1.7.9.5-1_amd64.deb) ...
Processing triggers for man-db ...
Setting up liberror-perl (0.17-1) ...
Setting up git-man (1:1.7.9.5-1) ...
Setting up git (1:1.7.9.5-1) ...
#

3. Download Cisco OpenStack installer scripts from github:

```
# cd /root && git clone -b havana
https://github.com/CiscoSystems/puppet_openstack_builder && cd
puppet_openstack_builder && git checkout h.2
```

Cloning into 'puppet_openstack_builder'...
remote: Counting objects: 3882, done.
remote: Compressing objects: 100% (1746/1746), done.
remote: Total 3882 (delta 2084), reused 3763 (delta 1998)
Receiving objects: 100% (3882/3882), 859.56 KiB | 383 KiB/s, done.
Resolving deltas: 100% (2084/2084), done.
Note: checking out 'h.2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b new_branch_name
```

HEAD is now at 2d294ee... Merge branch 'havana' into coi-development

4. Prior to running install script, it is important to populate critical information in the
~/.puppet_openstack_builder/data/hiera_data/user.common.yaml file. This file is heavily
commented, but most important information to be filled-in here is the proxy information and cobbler
node information (which would be same as build node). If your system is behind HTTP/HTTPS
proxy, then you must provide proxy information in the “proxy:” line. In addition, you can fill-in
various Networking related information at this point too. Keep in mind the following roles of
various interfaces:

- eth0: Public interface, accessible thru SSH from outside the pod
- eth1: Private / Data / External interface. This would be taken over by OVS.
- eth2: Storage replication interface; only applicable for storage nodes.

Complete content of user.common.yaml used in this solution is dumped in step 12, where Ceph
configuration is also covered.

5. Cisco OpenStack installer supports various installation scenario. “compressed_ha” scenario
provides OpenStack control plane high-availability with reduced number of hosts. “compressed_ha”
scenario requires 3 controller nodes that run all OpenStack control daemons, including Ceph-Mon
processes for high availability. Export right set of variables and start the installation script:

```
# unset http_proxy
# unset https_proxy
# export vendor=cisco
# export scenario=compressed_ha
# cd ~/.puppet_openstack_builder/install-scripts
#. /install.sh 2>&1 | tee install.log
+ set -e
+ set -e
+ export vendor=cisco
```
6. Run the following commands to populate the data model for puppet.

   # vi /etc/puppet/data/hiera_data/user.yaml
   # Set the hostname of the build node
   coe::base::build_node_name: ucs-openstack-buildnode
   # Set the hostname of the control node
   coe::base::controller_hostname: controller-node1
   # Set the IP address of the control node
   coe::base::controller_node_internal: 10.104.252.141

7. Define a few specific roles for the nodes. Edit the compressed_ha.yaml file located at
   /etc/puppet/data/scenarios and replace its content with following:

   # cat /etc/puppet/data/scenarios/compressed_ha.yaml
   #
   # 3 node compressed ha
   #
   roles:
   build:
     classes:
      - coe::base
     class_groups:
      - test_file
      - build
8. Replace the content of role_mappings.yaml file with just the build server information. The remaining of the role_mappings.yaml file would be updated by UCS integration scripts later.

```bash
# cat /etc/puppet/data/role_mappings.yaml
build-server: ucs-openstack-buildnode
```

9. Edit the build server related data in the build-server.yaml file as shown below:

```bash
# cat /etc/puppet/data/hiera_data/hostname/build-server.yaml
# set my puppet_master_address to be fqdn
puppet_master_address: "%{fqdn}"
cobbler_node_ip: '10.104.252.90'
node_subnet: '10.104.252.0'
node_netmask: '255.255.255.0'
node_gateway: '10.104.252.1'
admin_user: openstack-admin
password_crypted: $6$UfgWxrIv$Rea6HsrmC4FHv8oAaPFX5aejJxKEPafUYw6YKyseIrmm0I4rob/n3NgjJ3ITF25apmlEermX3rnN5e8x6j0
autostart_puppet: true
ucsm_port: 443
install_drive: /dev/sda
```

This file should be copied as the actual hostname.yaml. Run the following command to copy.

```bash
# cp /etc/puppet/data/hiera_data/hostname/build-server.yaml /etc/puppet/data/hostname/ucs-openstack-buildnode.yaml
```

The IP address and subnet in this file is the IP address and subnet of the build server. To generate a password encrypted, run the following command.
10. Run the following commands to setup Cobbler.

11. Setup cobbler configuration, which is located in the file:

```
# vi /etc/puppet/data/cobbler/cobbler.yaml
```

This file is divided in four main sections.

a. The first section "preseed", refers to which OpenStack repository that cobbler would use to install various modules. It should be as following:

```
presseed:
  repo: "http://openstack-repo.cisco.com/openstack/cisco havana-proposed main"
```
b. The second section is "profile"; which bare-metal operating system is to be installed on each server. "log_host" here refers to syslog server, which can be build node itself. "log_port" refers to UDP port 514 is for syslog server.

```bash
profile:
  name: "precise"
  arch: "x86_64"
  kopts: "log_port=514 \n  priority=critical \n  local=en_US \n  log_host=10.104.252.90 \n  netcfg/choose_interface=auto"
```

c. The third section "node_global" refers to common configuration across all the servers. In this case, Cisco UCS Manager managed server have commong configuration that goes under "node_global". Here, most of the data would be same as below for your deployment, except "power_pass", which is Cisco UCS Manager admin password and DNS name server IP address, which should be the Build Node IP address, as the FQDN of the OpenStack nodes may not be resolved by your organization's DNS server.

```bash
node-global:
  profile: "precise-x86_64"
  netboot_enabled: "1"
  power_type: "ucs"
  power_user: "admin"
  power_pass: "Nbv12345"
  kickstart: "/etc/cobbler/preseed/cisco-preseed"
  kopts: "netcfg/get_nameservers=10.104.252.90 \n  netcfg/confirn_static=true \n  netcfg/get_ipaddress=${eth0_ip-address} \n  netcfg/get_gateway=10.104.252.1 \n  netcfg/disable_autoconfig=true \n  netcfg/dhcp_options="Configure network manually" \n  partman-auto/disk=/dev/sda \n  netcfg/get_netmask=255.255.255.0 \n  netcfg/dhcp_failed=true"
```

d. The fourth section is information about each node. This can be the longest section in a typical deployment, however in case of UCS integration script, all controller nodes, compute nodes and storage nodes information would be populated by the script automatically. You need to populate only the Build Node information, such as hostname and IP address & MAC address of eth0 interface. "power_address" of Build Node can be any dummy IP address.

```bash
build-server:
  hostname: "ucs-openstack-buildnode.cisco.com"
  power_address: "192.168.2.101"
  interfaces:
    eth0:
      mac-address: "44:03:a7:29:57:93"
      dns-name: "ucs-openstack-buildnode.cisco.com"
      ip-address: "10.104.252.1"
      static: "0"
```

12. Configure Ceph for the storage. Edit the `/etc/puppet/data/hiera_data/user.common.yaml` file, where most of the configuration will remain same, except networking related configuration.

```bash
# cat /etc/puppet/data/hiera_data/user.common.yaml
#
#### Build Node Cobbler Information ######
# If you use an HTTP/HTTPS proxy to reach the apt repositories that # packages will be installed from during installation, uncomment this # setting and specify the correct proxy URL. If you do not use an # HTTP/HTTPS proxy, leave this setting commented out.
proxy: 'http://proxy-wsa.esl.cisco.com:80'
```
### Disk partitioning options

# The /var directory is where logfiles and instance data are stored on disk. If you wish to have /var on its own partition (considered a best practice), set enable_var to true.

enable_var: true

# The Cinder volume service can make use of unallocated space within the "cinder-volumes" volume group to create iSCSI volumes for export to instances. If you wish to leave free space for volumes and not preallocate the entire install drive, set enable_vol_space to true.

enable_vol_space: true

# Use the following two directives to set the size of the / and /var partitions, respectively. The var_part_size directive will be ignored if enable_var is not set to true above.

root_part_size: 65536

var_part_size: 432000

# This domain name will be the name your build and compute nodes use for the local DNS. It doesn’t have to be the name of your corporate DNS - a local DNS server on the build node will serve addresses in this domain - but if it is, you can also add entries for the nodes in your corporate DNS environment they will be usable *if* the above addresses are routeable from elsewhere in your network.

domain_name: cisco.com

############### NTP Configuration ###############

# Change this to the location of a time server or servers in your organization accessible to the build server. The build server will synchronize with this time server, and will in turn function as the time server for your OpenStack nodes.

ntp_servers:
  - ntp.esl.cisco.com

# The time zone that clocks should be set to. See /usr/share/zoneinfo for valid values, such as "UTC" and "US/Eastern".

time_zone: UTC

############### Node Addresses ###############

# Change the following to the short host name you have given your build node.
# This name should be in all lower case letters due to a Puppet limitation (refer to http://projects.puppetlabs.com/issues/1168).

build_node_name: ucs-openstack-buildnode

# Change the following to the short host name you have given your control node. This name should be in all lower case letters due to a Puppet limitation (refer to http://projects.puppetlabs.com/issues/1168).

controller_hostname: controller-node1

# The IP address to be used to connect to Horizon and external services on the control node. In the compressed_ha or full_ha scenarios, this will be an address to be configured as a VIP on the HAProxy load balancers, not the address of the control node itself.

controller_public_address: 10.104.252.150

# The protocol used to access API services on the control node. Can be 'http' or 'https'.

controller_public_protocol: 'http'

# The IP address used for internal communication with the control node. In the compressed_ha or full_ha scenarios, this will be an address to be configured as a VIP on the HAProxy load balancers, not the address of the control node itself.
controller_internal_address: 10.104.252.150

# The IP address used for management functions (such as monitoring)
# on the control node. In the compressed_ha or full_ha scenarios, this will
# be an address to be configured as a VIP on the HAProxy
# load balancers, not the address of the control node itself.
controller_admin_address: 10.104.252.150

# Control node interfaces.
# internal_ip be used for the ovs local_ip setting for GRE tunnels.

# This sets the IP for the private(internal) interface of controller nodes
# (which is predefined already in $controller_node_internal, and the internal #
# interface for compute nodes. It is generally also the IP address
# used in Cobbler node definitions.
internal_ip: "%(ipaddress_eth0)"

# The external_interface is used to provide a Layer2 path for
# the l3 agent external router interface. It is expected that
# this interface be attached to an upstream device that provides
# a L3 router interface, with the default router configuration
# assuming that the first non "network" address in the external
# network IP subnet will be used as the default forwarding path
# if no more specific host routes are added.
external_interface: eth1

# The public_interface will have an IP address reachable by
# all other nodes in the openstack cluster. This address will
# be used for API Access, for the Horizon UI, and as an endpoint
# for the default GRE tunnel mechanism used in the OVS network
# configuration.
public_interface: eth0

# The interface used for VM networking connectivity. This will usually
# be set to the same interface as public_interface.
private_interface: eth1

### Cobbler config
# The IP address of the node on which Cobbler will be installed and
# on which it will listen.
cobbler_node_ip: 10.104.252.90

# The subnet address of the subnet on which Cobbler should serve DHCP
# addresses.
node_subnet: '10.104.252.0'

# The netmask of the subnet on which Cobbler should serve DHCP addresses.
node_netmask: '255.255.255.0'

# The default gateway that should be provided to DHCP clients that acquire
# an address from Cobbler.
node_gateway: '10.104.252.1'

# The admin username and crypted password used to authenticate to Cobbler.
admin_user: openstack-admin
password_crypted: $6$UfgWxrIv$Rea6HsrmC4FHV8oAaPFX5aejJxKEpafUYwo6YKyseIrmmr0I4rob/n3NgjJEITP25apm1BermzX3rn5e8x6Jb0

# Cobbler can instruct nodes being provisioned to start a Puppet agent
# immediately upon bootup. This is generally desirable as it allows
# the node to immediately begin configuring itself upon bootup without
# further human intervention. However, it may be useful for debugging
# purposes to prevent Puppet from starting automatically upon bootup.
# If you want Puppet to run automatically on bootup, set this to true.  
# Otherwise, set it to false.
autostart_puppet: true

# If you are using Cisco UCS servers managed by UCSM, set the port on
# which Cobbler should connect to UCSM in order to power nodes off and on.
# If set to 443, the connection will use SSL, which is generally
# desirable and is usually enabled on UCS systems.
ucsm_port: 443

# The name of the hard drive on which Cobbler should install the operating
# system.
install_drive: /dev/sda

# Set to 1 to enable ipv6 route advertisement.  Otherwise, comment out
# this line or set it to 0.
#ipv6_ra: 1

# Uncomment this line and set it to true if you want to use bonded
# ethernet interfaces.
#interface_bonding: true

# The IP address on which vncserver proxyclient should listen.
# This should generally be an address that is accessible via
# horizon.  You can set it to an actual IP address (e.g. "192.168.1.1"),
# or use facter to get the IP address assigned to a particular interface.
nova::compute::vncserver_proxyclient_address: "${ipaddress_eth0}"

### The following are passwords and usernames used for
### individual services.  You may wish to change the passwords below
### in order to better secure your installation.
cinder_db_password: cinder_pass
glance_db_password: glance_pass
keystone_db_password: key_pass
nova_db_password: nova_pass
network_db_password: quantum_pass
database_root_password: mysql_pass

nova_service_password: nova_pass
cinder_service_password: cinder_pass
glance_service_password: glance_pass
nova_service_password: nova_pass
ceilometer_service_password: ceilometer_pass
admin_password: Cisco123
admin_token: keystone_admin_token
	network_service_password: quantum_pass
rpc_password: openstack_rabbit_password
metadata_shared_secret: metadata_shared_secret
database_root_password: mysql_pass
heat::engine::auth_encryption_key: 'notgood but just long enough i think'

# Set this parameter to use a single secret for the Horizon secret
# key, neutron agents, Nova API metadata proxies, swift hashes,etc.
# This prevents you from needing to specify individual secrets above,
# but has some security implications in that all services are using
# the same secret (creating more vulnerable services if it should be
# compromised).
#secret_key: secret

# Set this parameter to use a single password for all the services above.
# This prevents you from needing to specify individual passwords above,
# but has some security implications in that all services are using
# the same password (creating more vulnerable services if it should be
# compromised.
#password: password123

### Swift configuration
# The username used to authenticate to the Swift cluster.
swift_service_password: swift_pass

# The password hash used to authenticate to the Swift cluster.
swift_hash: super_secret_swift_hash

# The IP address used by Swift on the control node to communicate with
# other members of the Swift cluster. In the compressed_ha or full_ha
# scenarios, this will be the address to be configured as a VIP on
# the HAProxy load balancers, not the address of an individual Swift node.
swift_internal_address: 10.104.252.150

# The IP address which external entities will use to connect to Swift,
# including clients wishing to upload or retrieve objects. In the
# compressed_ha or full_ha scenarios, this will be the address to
# be configured as a VIP on the HAProxy load balancers, not the address
# of an individual Swift node.
swift_public_address: 10.104.252.150

# The IP address over which administrative traffic for the Swift
# cluster will flow. In the compressed_ha or full_ha
# scenarios, this will be the address to be configured as a VIP on
# the HAProxy load balancers, not the address of an individual Swift node.
swift_admin_address: 10.104.252.150

# The interface on which Swift will run data storage traffic.
# This should generally be a different interface than is used for
# management traffic to avoid congestion.
swift_storage_interface: eth2

# The IP address to be configured on the Swift storage interface.
swift_local_net_ip: "${ipaddress_eth2}"

# The netmask to be configured on the Swift storage interface.
swift_storage_netmask: 255.255.255.0

# The IP address of the Swift proxy server. This is the address which
# is used for management, and is often on a separate network from
# swift_local_net_ip.
swift_proxy_net_ip: "${ipaddress_eth0}" 

### Ceph configuration
# The name of the Ceph cluster to be deployed.
ceph_cluster_name: 'ceph'

# The FSID of the Ceph monitor node. This should take the form
# of a UUID.
ceph_monitors_fsid: 'e80afa94-a64c-486c-9e34-d55e85f26406'

# The shared secret used to connect to the Ceph monitors. This
# should be a crypted password.
ceph_monitors_secret: 'AQAJzNxR+PNRIRAA7yUp9hJiJdWZ3PVz242Xjlw=='

# The short hostname (e.g. 'ceph-mon01', not 'ceph-mon01.domain.com') of
# the initial members of the Ceph monitor set.
mon_initial_members: 'controller-node1'

# The short hostname (e.g. 'ceph-mon01', no 'ceph-mon01.domain.com') of
# the primary monitor node.
ceph_primary_mon: 'controller-node1'

# The IP address used to connect to the primary monitor node.
ceph_monitors_address: '10.104.252.141'

# Ceph will be deployed using the cephdeploy tool. This tool requires
# a username and password to authenticate.
ceph_deploy_user: 'cephdeploy'
ceph_deploy_password: '9jfd29k9kd9'

# The name of the network interface used to connect to Ceph nodes.
# This interface will be used to pass traffic between Ceph nodes.
ceph_cluster_interface: 'eth2'

# The subnet on which Ceph intra-cluster traffic will be passed.
ceph_cluster_network: '10.61.61.0/24'

# The interface on which entities that want to import data into or
# extract data from the cluster will connect.
ceph_public_interface: 'eth0'

# The subnet on which external entities will connect to the Ceph cluster.
ceph_public_network: '10.104.252.0/24'

# A list of kernel modules that should be added to /etc/modules
# and thus auto-loaded into the kernel at boot time. It is recommended
# that you include at least two modules: "8021q" is necessary for vlan
# tagging, and "vhost_net" increases networking performance of instances.
kernel_module_list:
  - 8021q
  - vhost_net

### The following four parameters are used only if you are configuring
### Ceph to be a backend for the Cinder volume service.

# Enable Cinder to use Ceph RBD as it's backend storage by uncommenting
# the line below. It can also be set to 'iscsi'.
cinder_backend: rbd

cinder_rbd_user: 'admin'

cinder_rbd_pool: 'volumes'

# The UUID of the shared virsh secret used by Cinder to
# authenticate to the Ceph cluster.
cinder_rbd_secret_uuid: 'e80afa94-a64c-486c-9e34-d55e85f26406'

### The following parameter is used only if you are deploying Ceph
### as a backend for the Glance image service.
# The name of the pool used to store glance images.

`glance_ceph_pool: 'images'`

### The following parameters relate to Neutron L4-L7 services.

# A boolean specifying whether to enable the Neutron Load Balancing as a Service agent.
`enable_lbaas: true`

# A boolean specifying whether to enable the Neutron Firewall as a Service feature.
`enable_fwaas: true`

# A boolean specifying whether to enable the Neutron VPN as a Service feature.
`enable_vpnaaS: true`

# An array of Neutron service plugins to enable.

`service_plugins:
  - 'neutron.services.loadbalancer.plugin.LoadBalancerPlugin'
  - 'neutron.services.firewall.fwaas_plugin.FirewallPlugin'
  - 'neutron.services.vpn.plugin.VPNDriverPlugin'

# A hash of Neutron services to enable GUI support for in Horizon.

`horizon_neutron_options:
  'enable_lb': true
  'enable_firewall': true
  'enable_vpn': true`

# A boolean stating whether to run a "neutron-db-manage" on the nodes running neutron-server after installing packages. In most cases this is not necessary and may cause problems if the database connection information is only located in the neutron.conf file rather than also being present in the Neutron plugin's conf file.

`neutron_sync_db: false`

### NOTE: If enable_ssl is true, Replace the following lines with valid SSL certs. To generate your own self signed certs refer to the instructions from the following url https://help.ubuntu.com/12.04/serverguide/certificates-and-security.html. After generating your certs, make sure the certs are copied to /etc/keystone/ssl/ on your control nodes and rerun puppet agent.

# SSL client certificate

`ssl_certfile: '/etc/keystone/ssl/certs/keystone.pem'`

# SSL certificate key

`ssl_keyfile: '/etc/keystone/ssl/private/keystonekey.pem'`

# SSL CA Cert

`ssl_ca_certs: '/etc/keystone/ssl/certs/ca.pem'`

# SSL CA Key

`ssl_ca_key: '/etc/keystone/ssl/private/cakey.pem'`

# SSL cert subject

`ssl_cert_subject: '/C=US/ST=Unset/L=Unset/O=Unset/CN=localhost'`
13. Configure one file for each OSD server and store as the hostname. A sample file is provided at `/etc/puppet/data/hiera_data/hostname/ceph01.yaml`. Copy the file to `/etc/puppet/data/hiera_data/hostname/<your-host-name>.yaml` for each Storage Node.

```bash
# cat /etc/puppet/data/hiera_data/hostname/storage-node1.yaml
# example for using disks on a certain server for Ceph OSDs
cephdeploy::osdwrapper::disks:
- sdb
- sdc
- sdd
- sde
- sdf
- sdg
- sdh
- sdi
- sdj
- sdk
- sdl
- sm
- sdn
- sdo
- sdp
- sdq
- sdr
- sds
- sdt
- sdu
- sdv
- sdw
- sdx
```

14. Configure Cinder and Glance to use Ceph for storage. Edit the `/etc/puppet/data/global_hiera_params/scenario/compressed_ha.yaml` file and change backend for Cinder and Glance to “rdb” for Ceph:

```yaml
cinder_backend: rdb
glance_backend: rdb
```

15. Install UCS python SDK and UCS integration scripts. Ensure that Python 2.7.3 (or later) is installed. Run the following command to check the Python version

```bash
# python -V
Python 2.7.3
```

a. Run the following commands to install necessary packages:

- python-daemon
- python-loggingx
- python-yaml
- python-jinja2
- python-passlib

```
# apt-get install python-daemon python-yaml python-jinja2 python-passlib python-loggingx
```

Note: The output may vary depending on which modules are already installed on your system.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-daemon is already the newest version.
python-jinja2 is already the newest version.
```
16. Run the following commands to install UCS Python SDK.

Note: UCS integration script uses north bound XML API to communicate with Cisco UCS Manager. You would need UCS Python SDK to invoke XML API on Cisco UCS Manager.

```bash
# wget https://communities.cisco.com/servlet/JiveServlet/download/36899-4-60058/UcsSdk-0.8.tar.gz
# tar zxf UcsSdk-0.8.tar.gz
# cd UcsSdk-0.8/
# python setup.py install
```

17. Download the UCS integration scripts from the following location and extract the zip file. You may need to download and install ‘unzip’ using “apt-get install unzip”

```bash
# unzip integration.zip
```
18. Edit ucs_config_template.conf file with appropriate information. Following is the configuration used for the reference architecture. You need to modify IP addresses, VLANs and port information.

MacPool {'Name': 'ucs-macs', 'From': '00:25:B5:60:0D:00', 'To': '00:25:B5:60:0D:FF'}

# ext-mgmt is used to assign ips for physical servers
IpPool {'Name': 'ext-mgmt', 'From': '10.65.121.204', 'To': '10.65.121.212', 'DefGw': '10.65.121.1', 'PrimDns': '172.29.74.154', 'SecDns': '172.29.74.155', 'Subnet': '255.255.255.0'}

# Storage LocalDiskConfigurationPolicy
StorageLocalDiskConfigPolicy {'Name': 'storage-nodes', 'Mode': 'no-raid', 'ProtectConfig': 'no'}
StorageLocalDiskConfigPolicy {'Name': 'os-compute-nodes', 'Mode': 'raid-striped-parity', 'ProtectConfig': 'no'}
StorageLocalDiskConfigPolicy {'Name': 'os-control-nodes', 'Mode': 'raid-striped-parity', 'ProtectConfig': 'no'}

# BootPolicy
LsBootPolicy {'Name': 'openstack-node', 'Purpose': 'operational', 'RebootOnUpdate': 'no', 'EnforceVnicName': 'no', 'vm-ro': {'Type': 'vm', 'Access': 'read-only', 'Order': '1'}, 'lan': {'Type': 'lan', 'Access': 'read-only', 'Prot': 'pxe', 'Order': '2', 'ImagePath': {'Type': 'primary', 'VnicName': 'eth0'}}, 'storage': {'Type': 'storage', 'Access': 'read-write', 'Order': '3'}, 'vm-rw': {'Type': 'vm', 'Access': 'read-write', 'Order': '4'}, 'TargetOrg': 'org-root', 'Descr': 'openstack Integration'}

# ServiceProfile Template
ServiceProfile {'Name': 'control-node', 'Type': 'updating-template', 'BootPolicyName': 'openstack-node', 'Uuid': 'derived', 'HostFwPolicyName': 'default', 'ScrubPolicyName': 'control-nodes', 'ExtIPPoolName': 'control-nodes-ip', 'LocalDiskPolicyName': 'os-control-nodes'}
ServiceProfile {'Name': 'compute-node', 'Type': 'updating-template', 'BootPolicyName': 'openstack-node', 'Uuid': 'derived', 'HostFwPolicyName': 'default', 'ScrubPolicyName': 'compute-nodes', 'ExtIPPoolName': 'compute-nodes-ip', 'LocalDiskPolicyName': 'os-compute-nodes'}
ServiceProfile {'Name': 'storage-node', 'Type': 'updating-template', 'BootPolicyName': 'openstack-node', 'Uuid': 'derived', 'HostFwPolicyName': 'default', 'ScrubPolicyName': 'storage-nodes', 'ExtIPPoolName': 'storage-nodes-ip', 'LocalDiskPolicyName': 'storage-nodes'}

LsRequirement {'SrcTemplDn': 'org-root/ls-control-node', 'Name': 'compute-pool', 'Qualifier': 'compute-pool'}
LsRequirement {'SrcTemplDn': 'org-root/ls-compute-node', 'Name': 'compute-pool', 'Qualifier': 'compute-pool'}
LsRequirement {'SrcTemplDn': 'org-root/ls-storage-node', 'Name': 'storage-pool', 'Qualifier': 'storage-pool'}

UplinkPort {'SwitchId': 'A', 'SlotId': '2', 'PortId': '1', 'AdminSpeed': '10gbps'}
UplinkPort {'SwitchId': 'A', 'SlotId': '2', 'PortId': '2', 'AdminSpeed': '10gbps'}
UplinkPort {'SwitchId': 'B', 'SlotId': '2', 'PortId': '1', 'AdminSpeed': '10gbps'}
UplinkPort {'SwitchId': 'B', 'SlotId': '2', 'PortId': '2', 'AdminSpeed': '10gbps'}

Vlan {'Name': 'Infra', 'SwitchId': 'dual', 'Id': '613', 'Sharing': 'none', 'DefaultNet': 'no'}
Vlan {'Name': 'Data', 'SwitchId': 'dual', 'Id': '60', 'Sharing': 'none', 'DefaultNet': 'no'}
Vlan {'Name': 'Storage', 'SwitchId': 'dual', 'Id': '90', 'Sharing': 'none', 'DefaultNet': 'no'}

# Compute nodes would have two VNICs, eth0 having Infra VLAN while eth1 having both Infra and Data VLANs
Vnic {'Name': 'eth0', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'A', 'LsServer': 'compute-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Data', 'SwitchId': 'B', 'LsServer': 'compute-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'B', 'LsServer': 'compute-node', 'DefaultNet': 'no'}

# Control nodes would have two VNICs, eth0 having Infra VLAN while eth1 having both Infra and Data VLANs
Vnic {'Name': 'eth0', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'A', 'LsServer': 'control-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Data', 'SwitchId': 'B', 'LsServer': 'control-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'B', 'LsServer': 'control-node', 'DefaultNet': 'no'}

# Storage nodes would have three VNICs, eth0 having Infra VLAN, eth1 having both Infra and Data VLANs, and eth2 would have storage VLAN used for replication
Vnic {'Name': 'eth0', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'A', 'LsServer': 'control-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Data', 'SwitchId': 'B', 'LsServer': 'control-node', 'DefaultNet': 'yes'}
Vnic {'Name': 'eth1', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Infra', 'SwitchId': 'B', 'LsServer': 'control-node', 'DefaultNet': 'no'}
Vnic {'Name': 'eth2', 'IdentPoolName': 'ucs-macs', 'VlanName': 'Storage', 'SwitchId': 'B', 'LsServer': 'control-node', 'DefaultNet': 'yes'}

ServerPool {'Name': 'compute-pool'}
ServerPool {'Name': 'control-pool'}
ServerPool {'Name': 'storage-pool'}

# Any server with less than 8 TB of disk capacity would be used for compute and control nodes
RackServerQualifier {'Name': 'control-pool', 'MinCap': '1', 'MaxCap': '8388608'}
RackServerQualifier {'Name': 'compute-pool', 'MinCap': '1', 'MaxCap': '8388608'}

# Any server with more than 8 TB of disk capacity would be used for storage nodes
RackServerQualifier {'Name': 'storage-pool', 'MinCap': '8388608', 'MaxCap': 'unknown'}
ServerPoolingPolicy {'Name': 'compute-nodes', 'Qualifier': 'compute-pool', 'PoolDn': 'org-root/compute-pool-compute-pool'}
ServerPoolingPolicy {'Name': 'control-nodes', 'Qualifier': 'control-pool', 'PoolDn': 'org-root/compute-pool-control-pool'}
ServerPoolingPolicy {'Name': 'storage-nodes', 'Qualifier': 'storage-pool', 'PoolDn': 'org-root/compute-pool-storage-pool'}
ScrubPolicy {'Name':'compute-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}
ScrubPolicy {'Name':'control-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}
ScrubPolicy {'Name':'storage-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}

#Create service-profile Instances
ServiceProfileInstance {'SrcTempl':'compute-node', 'NamePrefix':'os-compute', 'NumberOf':'3', "TargetOrg":"org-root"}
ServiceProfileInstance {'SrcTempl':'control-node', 'NamePrefix':'os-control', 'NumberOf':'3', "TargetOrg":"org-root"}
ServiceProfileInstance {'SrcTempl':'storage-node', 'NamePrefix':'os-storage', 'NumberOf':'3', "TargetOrg":"org-root"}

# server-prots config goes here.
ServerPort {'PortId':'1', 'SlotId':'1', 'SwitchId':'A'}
ServerPort {'PortId':'1', 'SlotId':'1', 'SwitchId':'B'}
ServerPort {'PortId':'2', 'SlotId':'1', 'SwitchId':'A'}
ServerPort {'PortId':'2', 'SlotId':'1', 'SwitchId':'B'}

19. iplist.yaml file plays a crucial role in mapping service profiles, hostnames, IP address and the role of the node. Its syntax is as follows:

iplist:
  <service-profile-name>:
    name: <hostname>
    ip: <ipv4-address>
    type: <node-type>
    ...
    ...

# Node types is the list of all possible node types in a given deployment scenario
nodetypes:
  <nodetype1>
  <nodetype2>
  ...

The iplist.yaml file used in this architecture is as follows:

# cat /root/ucs_int/hiera/iplist.yaml
iplist:
  os-control1:
    name: controller-node1
    ip: 10.104.252.141
    type: compressed_ha_cephmon
  os-control2:
    name: controller-node2
    ip: 10.104.252.142
    type: compressed_ha_cephmon
  os-control3:
    name: controller-node3
    ip: 10.104.252.143
    type: compressed_ha_cephmon
  os-compute1:
    name: compute-node1
    ip: 10.104.252.144
    type: compute
  os-compute2:
    name: compute-node2
    ip: 10.104.252.145
    type: compute
  os-compute3:
    name: compute-node3
    ip: 10.104.252.146
    type: compute
20. Run the following command to check the initial Cisco UCS Manager state

```
UCS-POD-A# show cluster state
Cluster Id: 0xec91409a491011e2-0xb7a4547feeaa1564
A: UP, PRIMARY
B: UP, SUBORDINATE

HA NOT READY
No device connected to this Fabric Interconnect
```

21. If you have more than one 10-G Ethernet links connected between Fabric Interconnects and Fabric Extenders, you would need to manually change chassis discovery policy on the Cisco UCS Manager to take advantage of maximum bandwidth. You can do so by using either Cisco UCS Manager GUI or CLI. Following is the example CLI to modify chassis-discovery policy. It is recommended to use port-channel between Fabric Interconnects and Fabric Extenders for better high-availability and load-balancing.

```
# ssh admin@ucsm-vip
Cisco UCS 6200 Series Fabric Interconnect
Using keyboard-interactive authentication.
Password:
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (c) 2009, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php
UCS-POD-A# scope org /
UCS-POD-A /org # scope chassis-disc-policy
UCS-POD-A /org/chassis-disc-policy # set action 2-link
UCS-POD-A /org/chassis-disc-policy* # set link-aggregation-pref port-channel
UCS-POD-A /org/chassis-disc-policy* # commit-buffer
UCS-POD-A /org/chassis-disc-policy#
```

22. Run the python script to configure Cisco UCS Manager. The script would prompt for the password for admin account, and configure Cisco UCS Manager.

```
# python ucs_integration.py --ucsm 10.65.121.228 --ucsm_user admin --config-file ./ucs_conf_template.conf 2>&1 | tee ucs_integration.log
```
Password:

UCS-POD ===> <?xml version="1.0" ?>
<configResolveClass classId="orgOrg"
cookie="1392278575/47fe7a-1281-489d-ab1b-fa366bc574e"
inHierarchical="false">
<inFilter>
<eq class="orgOrg" property="dn" value="org-root"/>
</inFilter></configResolveClass>

UCS-POD <==== <configResolveClass
cookie="1392278575/47fe7a-1281-489d-ab1b-fa366bc574e" response="yes"
classId="orgOrg" descr="" dn="org-root" fltAggr="524288"
level="root" name="root" permAccess="yes"/>
</configResolveClass>

UCS-POD ===> <?xml version="1.0" ?>
<configResolveClass classId="storageLocalDiskConfigPolicy"
inHierarchical="false">
<inFilter>
<eq class="storageLocalDiskConfigPolicy" property="dn" value="org-root/local-disk-config-storage-nodes"/>
</inFilter></configResolveClass>

UCS-POD <==== <configResolveClass
cookie="1392278575/47fe7a-1281-489d-ab1b-fa366bc574e" response="yes"
classId="storageLocalDiskConfigPolicy" descr="" dn="org-root/local-disk-config-storage-nodes""/>
</configResolveClass>

UCS-POD ===> <?xml version="1.0" ?>
<configResolveClass classId="fabricDceSwSrv"
inHierarchical="false">
<inFilter>
<eq class="fabricDceSwSrv" property="dn" value="fabric/server/sw-B"
locale="internal" name="" transport="dce" type="lan"/>
</inFilter></configResolveClass>

UCS-POD <==== <configResolveClass
cookie="1392270558/1051b190-bf71-42dd-aada-8d1bc5f9d520" response="yes"
classId="fabricDceSwSrv" descr="" dn="fabric/server/sw-B" id="B"
locale="internal" name="" transport="dce" type="lan"/>
</configResolveClass>

UCS-POD ===> <?xml version="1.0" ?>
<configResolveClass classId="fabricDceSwSrvEp"
inHierarchical="false">
<inFilter>
<eq class="fabricDceSwSrvEp" property="dn" value="fabric/server/sw-B/slot-1-port-2"
locale="internal" name="" transport="ether" type="lan"/>
</inFilter></configResolveClass>

UCS-POD <==== <configResolveClass
cookie="1392270558/1051b190-bf71-42dd-aada-8d1bc5f9d520" response="yes"
classId="fabricDceSwSrvEp" descr="" dn="fabric/server/sw-B/slot-1-port-2" id="2"
locale="internal" name="" transport="ether" type="lan"/>
</configResolveClass>

UCS-POD ===> <?xml version="1.0" ?>
<configConfMos
cookie="1392270558/1051b190-bf71-42dd-aada-8d1bc5f9d520"
inHierarchical="false">
<inConfigs>
<pair key="fabric/server/sw-B/slot-1-port-2">
<fabricDceSwSrvEp adminState="enabled"
dn="fabric/server/sw-B/slot-1-port-2" portId="2" slotId="1"
status="created"/>
</pair>
</inConfigs></configConfMos>

UCS-POD <==== <configConfMos
cookie="1392270558/1051b190-bf71-42dd-aada-8d1bc5f9d520" response="yes"
classId="fabricDceSwSrvEp" descr="" dn="fabric/server/sw-B/slot-1-port-2" id="2"
locale="internal" name="" transport="ether" type="lan"/>
</configConfMos>

# Once the script exits and gives control back to shell prompt, a python deamon would be running in
back ground to listen for Cisco UCS Manager compute discovery and association events. You can
check this out by issuing following command:

# ps -ef | grep integration
root 15039 1 12 14:10 ? 00:03:05 python ucs_integration.py --ucsm
10.65.121.228 --ucsm_user admin --config-file ./ucs_conf_template.conf
Validating Ubuntu OpenStack on UCS Platform

This section provides a list of items that should be reviewed once the solution has been configured. The goal of this section is to verify the configuration and functionality of specific aspects of the solution, and ensure that the configuration supports core availability requirements.

1. After you run the UCS integration scripts, Cisco UCS Manager would be configured for server side connectivity. You can see that C-series servers are discovered on Cisco UCS Manager, by clicking on the Equipment > Rack Mounts > Servers as shown in Figure 14. The servers will be in Discovery overall status for a while till the deep discovery finishes.

![Figure 14 - Discovered Cisco UCS Rack-Mount Servers](image)

2. In the Servers tab, and expand Servers > Service Profiles > root on the left hand side, you can see that the UCS integration script has created 9 service profiles, 3 for controller nodes, 3 for compute nodes and 3 for storage nodes as shown in Figure 15.
3. These service profiles are associated with server pools, and as servers are still in the process of discovery, the server pools are not populated. You can check the detailed status of service profiles by clicking on individual service profile, and would observe the status as shown in Figure 16.
4. Eventually, when servers are fully discovered, they would be put in to appropriate server pools. Our qualification criteria is - any server with more than 6TB local disks storage would be classified as storage pool, and any server with less than 6TB storage would be classified as compute and control pools. So, 3 x C240 M3 servers are populated in Storage pool and 6 x C220 M3 servers are populated in both control and compute pools. It is acceptable to have same server part of multiple pools. Once a server is associated to a service profile, it will not be available in all the pools. The default server pool will remain empty, as there is no qualification criteria attached to it. You can look at these details by expanding Servers > Pools > root, and clicking on Server Pools as shown in Figure 17.
5. Once all the servers are discovered and classified in to appropriate server pools, you would see that all the service profiles would start associating and eventually, all the 9 service profiles would be associated to various rack servers as shown in Figure 18.

![Figure 18 Service Profiles Associated to Various Rack Servers](image)

**Note** To see the final service profiles associated status relaunch Cisco UCS Manager a few minutes after running the UCS integration scripts.

6. Once service profiles are associated, the Build Node would continue Cobbler based bare metal provisioning, where the servers would be booted from the network and correct Ubuntu image would be installed on them. After installation of the Operating System, Puppet would push right set of OpenStack binaries and scripts on the nodes, depending on their roles. You can see all these by launching the KVM console of the server by selecting a particular service profile, right click and select **KVM Console** from the menu as shown in Figure 19.
Once various OpenStack components are installed on various nodes, especially the Controller nodes, you can login to the Horizon dashboard using the Virtual IP address of the controller node(s). Managing OpenStack cluster is beyond the scope of this CVD.
Troubleshooting

To manually prepare a node (other than build node) in OpenStack cluster, you can execute following set of commands:

```
apt-get install -y git
cd /root
  git clone https://github.com/CiscoSystems/puppet_openstack_builder
cd puppet_openstack_builder
  git checkout h.2
  cd install-scripts
  export build_server_ip=<build-node-IP>
  bash setup.sh
  puppet agent -td --server=<Build-Node-FQDN> --pluginsync
```

All the configuration of OpenStack cluster is stored in various .yaml files. Most of these files are heavily documented. In case of any deployment issues, ensure that values provided in these .yaml files are correct and consistent. Following is the order in which various .yaml files are parsed. The files are relative to path /etc/puppet/data/hiera_data.

- hostname/%{hostname}
- "client/{{clientcert}}"
- user
- jenkins
- vendor/cisco_coi_user_..{%{scenario}}
- user..{%{scenario}}
- vendor/cisco_coi_user.common
- user.common
- vendor/osfamily/cisco_coi_%{osfamily}
- osfamily/%{osfamily}
- enable_ha/%{enable_ha}
- install_tempest/%{install_tempest}
- cinder_backend/${cinder_backend}
- glance_backend/${glance_backend}
- rpc_type/${rpc_type}
- db_type/${db_type}
- tenant_network_type/${tenant_network_type}
- network_type/${network_type}
- network_plugin/${network_plugin}
- password_management/${password_management}
- scenario/${scenario}
- vendor/cisco_coi_common
- common

If you change any data in these files, you need to reapply puppet data model using following command:

```
puppet apply -v -d /etc/puppet/manifests/site.pp
```

After this, you would need to re-run following command on the nodes that are affected by the change in configuration:

```
export build_server_ip=<build-node-IP>
puppet agent -td --server=<Build-Node-FQDN> --pluginsync
```

### Post Install Checklist

The following configuration items are critical to functionality of the solution, and should be verified prior to deployment into production.

- Create a test virtual machine that accesses the datastore and is able to do read/write operations. Perform the virtual machine migration to a different compute node.
- During the live migration of the virtual machine, have a continuous ping to default gateway and make sure that network connectivity is maintained during and after the migration.

### Verify the Redundancy of the Solution Components

Following redundancy checks were performed at the Cisco lab to verify solution robustness. A continuous ping from VM to VM, and VM to outside network should not show significant failures (one or two ping drops might be observed at times, such as Fabric Interconnect reboot). Also, all the data-stores must be visible and accessible from all the hosts at all the time.

1. Administratively shutdown one of the two server ports connected to the Fabric Extender A. Make sure that connectivity is not affected. Upon administratively enabling the shutdown port, the traffic should be rebalanced. This can be validated by clearing interface counters and showing the counters after forwarding some data from virtual machines.
2. Administratively shutdown both server ports connected to Fabric Extender A. UCS VIC fabric failover should kick-in, and compute nodes should be able to use fabric B in this case.
3. Repeat steps 1 and 2. Make sure that storage is still available from all the compute nodes. Upon administratively enabling the shutdown port, the traffic should be rebalanced.
4. Reboot one of the two Fabric Interconnects while storage and network access from the compute nodes and VMs are going on. The switch reboot should not affect the operations of storage and network access from the VMs. Upon rebooting the Fabric Interconnect, the network access load should be rebalanced across the two fabrics.
5. Fully load all the virtual machines of the solution following the N+1 HA guidelines mentioned before. Choose one of the compute nodes and migrate all the VMs running on that node to other active nodes. No VM should lose any network or storage accessibility during or after the migration. Shutdown the compute node where no VMs are running.
6. Reboot one of the two storage nodes. All the VMs must be able to have storage access during storage node reboot.

7. Remove the disk 0 where operating system is installed on the Storage Node. This would bring down the entire storage node. Ceph based replication of the VM data should prevent any data loss from VM instances’ points of view. Decommission the server where the disk was removed. Replace it by a different disk and recommission the server. The new server should be discovered, associated and installed - and Ceph cluster should be eventually restored on the server again

Bill of Material

Table 6 gives details of the components used in the CVD for 100 virtual machines configuration.

Table 6  Component Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Part #</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 x UCS C220 M3 rack servers (1 x Build Node, 3 x Controller Nodes, 3 x Compute Nodes)</td>
<td>UCSC-C220-M3S</td>
</tr>
<tr>
<td>3 x UCS C240 M3 rack servers</td>
<td>UCSC-C240-M3S</td>
</tr>
<tr>
<td>CPU (2 per server)</td>
<td>UCS-CPU-E52660B</td>
</tr>
<tr>
<td>Memory for rack servers (8 per server)</td>
<td>UCS-MR-1X162RY-A</td>
</tr>
<tr>
<td>Cisco UCS 1225 VIC adapter (1 per server)</td>
<td>UCSC-PCIE-CSC-02</td>
</tr>
<tr>
<td>Cisco UCS LSI 9271-i Mega RAID controller card (1 per C240 M3)</td>
<td>UCS-RAID-9266CV</td>
</tr>
<tr>
<td>Hard disks for C240 M3 rack servers (24 per server)</td>
<td>A03-D1TBSATA</td>
</tr>
<tr>
<td>Cisco UCS 2008M-8i Mezzanine Storage Controller Cards</td>
<td>UCSC-RAID-11-C220</td>
</tr>
<tr>
<td>Hard disks for C220 M3 rack servers (4 per server)</td>
<td>UCS-HDD300G12F105</td>
</tr>
<tr>
<td>UCS 6248UP Fabric Interconnects (2)</td>
<td>UCS-FI-6248UP</td>
</tr>
<tr>
<td>UCS 2232PP Fabric Extenders (2)</td>
<td>N2K-C2232PP-10GE</td>
</tr>
<tr>
<td>Cisco Nexus 6001 Switches (2)</td>
<td>N6K-C6001-64P</td>
</tr>
<tr>
<td>10 Gbps SFP+ multifiber mode</td>
<td>SFP-10G-SR</td>
</tr>
</tbody>
</table>

For more information about the part numbers and options available for customization, refer:


Customer Configuration Data Sheet

Before you start the configuration, gather some customer-specific network and host configuration information. Table 7 to Table 11 provide information on assembling the required network and host address, numbering, and naming information. This worksheet can also be used as a “leave behind” document for future reference.

Table 7  Common Server information

<table>
<thead>
<tr>
<th>Server Name</th>
<th>Purpose</th>
<th>Primary IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DNS Primary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DNS Secondary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NTP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HTTP/HTTPS/FTP proxy (optional)</td>
<td></td>
</tr>
</tbody>
</table>

Table 8  OpenStack Node information

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Purpose</th>
<th>Management IP</th>
<th>Private IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Node</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Controller node 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller node 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller node 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute node 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute node 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute node 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage node 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage node 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage node 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9  Network infrastructure information

<table>
<thead>
<tr>
<th>Description</th>
<th>IP</th>
<th>Subnet Mask</th>
<th>Default Gateway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cisco UCS Manager Virtual IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCS Fabric Interconnect A address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCS Fabric Interconnect B address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6k-A Management IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6k-B Management IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Alternative HA configuration for Storage Nodes Operating System

For storage nodes, we have 24 local hard drives. We use first two disks to install Ubuntu 12.04.04 and remaining 22 disks would be exposed to OpenStack Ceph module to provide highly available block and object storage to VMs instantiated on the compute nodes. First of all, make sure that the Cisco UCS Manager local disk policy is set to “any-configuration” in the Build Node as shown in the following line of ucs_config_template.conf:

```bash
# Storage LocalDiskConfigurationPolicy
StorageLocalDiskConfigPolicy { 'Name' : 'storage-nodes', 'Mode' : 'any-configuration', 'ProtectConfig' : 'no'}
```

After altering that critical configuration, use following manual steps to configure each storage node.

1. Log in to the Cisco UCS Manager application.
2. Click the Equipment tab.
3. Click on one of the Storage-Node-x service profile and click on the KVM Console link on the right hand side as shown in Figure 21.

### Table 10 VLAN information

<table>
<thead>
<tr>
<th>Name</th>
<th>Network Purpose</th>
<th>VLAN ID</th>
<th>Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infra</td>
<td>Virtual Machine Networking</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Data VLAN for private network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>Storage replication VLAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tenant1</td>
<td>VLAN for tenant 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tenant2</td>
<td>VLAN for tenant 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 11 Service accounts

<table>
<thead>
<tr>
<th>Account</th>
<th>Purpose</th>
<th>Password (optional, secure appropriately)</th>
</tr>
</thead>
<tbody>
<tr>
<td>openstack-admin</td>
<td>Ubuntu host access, sudoer</td>
<td></td>
</tr>
<tr>
<td>admin</td>
<td>Cisco UCS Manager administrator</td>
<td></td>
</tr>
<tr>
<td>admin</td>
<td>N6k administrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OpenStack administrator</td>
<td></td>
</tr>
</tbody>
</table>
3. When server is power cycling or towards the end of service profile association, you would see following screen shot after the POST. When <Ctrl><H> for WebBIOS message appears, press Control + H on keyboard as shown in Figure 22.
4. Click **Start** to begin the WebBIOS configuration wizard as shown in Figure 23.

**Figure 23 WebBIOS Configuration Wizard**

5. Click **Clear** to erase all existing configuration as shown in Figure 24.
6. Click **Yes** on the confirmation page as shown in Figure 25.

7. Click **Configuration Wizard** to configure all the disks as shown in Figure 26.
Figure 26 Configure Disks

8. Click **New Configuration** and then click **Next** as shown in Figure 27.
Figure 27  
Select Configuration Type

Configuration Wizard guides you through the steps for configuring the MegaRAID system easily and efficiently. The steps are as follows:
1. Drive Group definitions  Group drives into Drive Groups.
2. Virtual Drive definitions Define virtual drives using those drive groups.
3. Configuration Preview  Preview configuration before it is saved.

Please choose appropriate configuration type:
- Clear Configuration  Allows you to clear existing configuration only.
- New Configuration  Clears the existing configuration. If you have any existing data in the earlier defined drives, the data will be lost.
- Add Configuration  Retains the old configuration and then adds new drives to the configuration. This is the safest operation as it does not result in any data loss.

9. Click Yes on the confirmation page as shown in Figure 28.

Figure 28  
New Configuration - Confirmation

You have chosen to clear the configuration. This will destroy all virtual drives. All data on all virtual drives will be lost.

Are you sure you want to clear the configuration?

10. Select Manual Configuration and click Next as shown in Figure 29.
11. Select first two SAS disks and click **Add to Array** to add the disks to the Drive Group0 as shown in Figure 30.
12. Once the disks are added to the drive group, click **Accept DG** as shown in Figure 31.
13. Select the disks from the list of available drives, and click **Add To Array** to add the disks to the drive groups on right hand side as shown in Figure 32.
14. Accept the drive group, repeat the steps for all the Unassigned drives on the host as shown in Figure 33.
15. Once all the drives are added to the drive groups, click Next as shown in Figure 34.
16. From the Array With Free Space list, click on Add to SPAN as shown in Figure 35.
17. Click **Next** once the Drive Group is added to the SPAN list as shown in Figure 36.
18. For the first two disks, use RAID 1 and click **Update Size** as shown in Figure 36.
19. Once size is updated, click **Accept** at the bottom of the window as shown in **Figure 37**.
20. Click Yes on the confirmation page as shown in Figure 38.

21. Click Back to select more drive groups from the previous page as shown in Figure 39.
22. From the previous page, repeat steps 19 and 20 to select next drive group and add to SPAN and click Next. For all next drive groups with single drive, select “RAID 0” and click Update Size and click Accept. Click Yes on the confirmation page. Repeat steps 24 and 25 for all the remaining drives on the system.
23. Once all the disks are configured in RAID0 drive group (except first two disks, which are in RAID1), click **Next** as shown in Figure 42.
24. Click **Accept** the newly defined configuration as shown in Figure 43.
25. Click Yes twice to save the configuration as shown in Figure 44.

26. Click Yes to delete all data on the new Virtual Drives as shown in Figure 45.
27. Select all the virtual disks, click **Fast Initialize** and then click **Go** as shown in Figure 46.

28. Click **Yes** on the warning message as shown in Figure 47.
29. For the first virtual disk (VD0), select Set Boot Drive and click Go as shown in Figure 48.

30. Click Home and exit the configuration as shown in Figure 49.
31. Click Yes on the confirmation page as shown in Figure 50.

32. Repeat these steps for the second storage node too. At this point, all the storage and compute nodes are ready for OS installation.

References

- Cisco UCS:
- Cisco Cisco UCS Manager 2.1 configuration guides:
• CLI:  

• GUI:  

• Ubuntu wiki:  
  http://wiki.ubuntu.com/

• Ceph Installation and Configuration  
  http://ceph.com/docs/master/install/rpm/