Technical Report

Highly Available OpenStack Deployments Built on NetApp Storage Systems

Solution Design

Bob Callaway, Greg Loughmiller, NetApp

May 2015 | TR-4323-DESIGN

Abstract

This document describes the solution design for a highly available deployment of OpenStack built on NetApp® storage systems. It includes architectural considerations, technology choices, and best practices for leveraging the functional and nonfunctional capabilities of NetApp storage systems toward an OpenStack deployment that provides self-service infrastructure-as-a-service (IaaS) capabilities to end users. It is based on the Kilo release of OpenStack (April 2015).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Solution Overview

Industry trends indicate a vast data center transformation toward an IT-as-a-service (ITaaS) paradigm. Enterprise customers and service providers are transforming shared infrastructures that heavily leverage virtualized environments into self-service cloud computing environments that increase business agility and enable rapid cycles of innovation.

OpenStack has emerged as the leading open-source infrastructure-as-a-service (IaaS) platform that aims to provide high-quality infrastructure and platform services through vendor-agnostic application programming interfaces (APIs). The services can be leveraged independently or combined by application developers to create composite applications for deployment in public, private, or hybrid topologies. This document describes the architecture of the core OpenStack services on top of NetApp storage in a highly available framework, independent of the particular OpenStack distribution chosen.

NetApp began contributing code in 2011 and joined the OpenStack Foundation as a charter member in August 2012. As a Gold Member, NetApp is pleased to be taking an active leadership role in the OpenStack community.

1.1 Target Audience

The target audience for this document includes the following groups:

- **Technical decision makers.** This document describes the value of using NetApp storage solutions with OpenStack cloud services to create a cloud environment that enterprises and service providers can use to meet their respective needs.
- **Enterprise and service provider cloud architects.** NetApp storage provides an enterprise-class underpinning for OpenStack-based clouds. This document describes key design considerations and best practices for deploying OpenStack in a highly available and scalable manner.
- **OpenStack community members and NetApp partners and implementers.** It is important for the OpenStack community and NetApp partners to understand the solution architecture of OpenStack on NetApp storage in order to meet and exceed customer requirements and expectations for their cloud solutions.

1.2 Solution Technology

This section describes OpenStack services and the NetApp storage solutions that can be used with them.

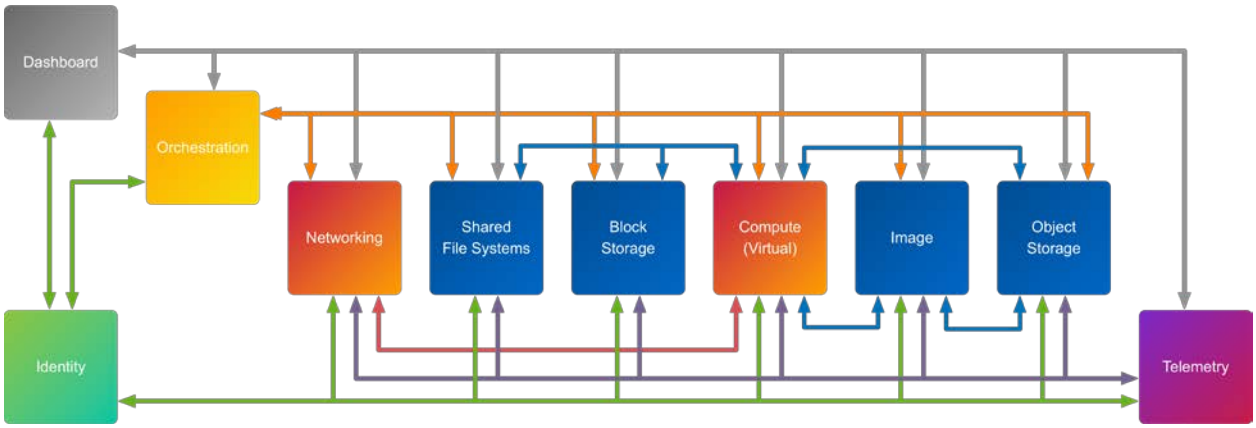
OpenStack

OpenStack implements services for establishing IaaS released under the Apache 2.0 open-source license. The project is managed by the OpenStack Foundation, a nonprofit corporate entity established in 2012 to promote, protect, and empower OpenStack software and its community.

This technology consists of a series of modular projects that control large pools of processing, storage, and networking resources throughout a data center, all managed through a single dashboard that gives administrators control while empowering users with self-service provisioning of resources.

Figure 1 shows the key OpenStack services.

Figure 1) High-level architecture of OpenStack services.



OpenStack Compute Service (Nova)

OpenStack enables enterprises and service providers to offer on-demand compute resources by provisioning and managing large networks of virtual machines (VMs). Compute resources are accessible through APIs for developers building cloud applications and through web interfaces for administrators and users. The OpenStack Compute Service (Nova) architecture is designed to scale horizontally on standard hardware. Nova is architected to avoid inherent proprietary hardware or software requirements and with the ability to integrate with existing systems and third-party technologies. It is designed to manage and automate pools of compute resources and can work with widely available virtualization technologies as well as with bare-metal and high-performance computing configurations.

OpenStack Block Storage Service (Cinder)

OpenStack Block Storage (Cinder) provides persistent block devices mapped to OpenStack compute instances (which are otherwise assumed to be ephemeral). Cinder manages the creation, attaching, and detaching of the block devices to instances. It also optionally supports instance booting and provides mechanisms for creating Cinder snapshots. Although fully integrated with Nova and Horizon, it can also be used independently from OpenStack to provide a standardized abstraction for block storage provisioning.

OpenStack Object Storage Service (Swift)

OpenStack Object Storage (Swift) provides a fully distributed, scale-out, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving, and data retention. Object Storage does not present a traditional file system, but rather a distributed storage system for static data such as VM images, photo storage, e-mail storage, backups, and archives. The Swift API, in a manner similar to Cloud Data Management Interface (CDMI), proposes an open standard for cloud storage. It can also function as an alternative endpoint for Amazon Web Services (AWS) Simple Storage Service (S3) and as a CDMI server through the use of add-on components.

OpenStack File Share Service (Manila)

OpenStack File Share Service (Manila) provides management of shared or distributed file systems such as Network File System (NFS) and Common Internet File System (CIFS). Manila provisions and coordinates access to persistent file-based storage for use by Nova instances or other external requestors. Like many other OpenStack services, Manila has a pluggable provider architecture that facilitates the creation and management of shared file systems, as well as a pluggable network integration model that supports all of the major OpenStack networking services. Manila became a formally incubated OpenStack project during the Juno release cycle and is generally regarded as available for production use in the Kilo release.

OpenStack Dashboard (Horizon)

The OpenStack Dashboard (Horizon) offers administrators and users a graphical interface to access, provision, and automate cloud-based resources. The extensible design makes it easy to plug in and use third-party products and services such as billing, monitoring, and additional management tools. The dashboard can also be made brand specific for service providers and other enterprises that require customization. The dashboard is one of several ways to interact with OpenStack resources. Developers can automate access or build tools to manage their resources by using the native OpenStack API or the EC2 compatibility API. The dashboard offers users a self-service portal for provisioning their own resources within the limits set by administrators.

OpenStack Identity Service (Keystone)

OpenStack Identity Service (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system (OS), and it can integrate with existing backend directory services such as Microsoft Active Directory (AD) and Lightweight Directory Access Protocol (LDAP). It supports multiple forms of authentication, including standard user name and password credentials, token-based systems, and AWS-style logins. In addition, the catalog lists all of the services deployed in an OpenStack cloud that can be queried in a single registry. Users and third-party tools can programmatically determine which resources they can access. Keystone enables:

- Configuration of centralized policies across users and systems
- Creation of users and tenants and definition of permissions for compute, storage, and networking resources by using role-based access control features
- Integration with existing directories, allowing a single source of identity information
- Listing of services that users can access and the ability to make API requests or log in to the dashboard to create resources to be owned by the user's account

OpenStack Image Service (Glance)

OpenStack Image Service (Glance) provides discovery, registration, and delivery services for disk and server images. The capability to copy a server image and immediately store it away is a powerful feature of OpenStack. When multiple servers are being provisioned, a stored image can be used as a template to get new servers up and running more quickly and consistently than by installing a server OS and individually configuring additional services. You can also use Glance to store and catalog an unlimited number of backups. It can store disk and server images in a variety of backends, including NFS and Object Storage. The Glance API provides a standard Representational State Transfer (REST) interface for querying information about disk images and allows clients to download the images to serve as the basis for new instances. A multiformat image registry allows uploads of private and public images in a variety of popular formats.

OpenStack Network Service (Neutron)

OpenStack Network Service (Neutron) is a pluggable, scalable, and API-driven system for managing networks and IP addresses. Like other aspects of the cloud OS, it can be used by administrators and users to increase the value of existing data center assets. Neutron prevents the network from becoming a bottleneck or limiting factor in a cloud deployment and provides users with self-service capability for their network configurations. The pluggable backend architecture lets users take advantage of basic commodity gear or advanced networking services from supported vendors. Administrators can take advantage of software-defined networking (SDN) technology such as OpenFlow to allow high levels of multi-tenancy and massive scale. Neutron has an extension framework for deploying and managing additional network services, such as intrusion detection systems (IDSs), load balancing, firewalls, and virtual private networks (VPNs).

OpenStack Telemetry (Ceilometer)

OpenStack Telemetry (Ceilometer) provides a common infrastructure for collecting usage and performance measurements in an OpenStack cloud. Its primary initial targets are monitoring and metering, but the framework can be expanded to collect data for other needs. Ceilometer was promoted from incubation status to an integrated component of OpenStack in the Grizzly release (April 2013).

OpenStack Orchestration (Heat)

OpenStack Orchestration (Heat) implements a service to orchestrate multiple composite cloud applications by using the AWS CloudFormation template format, through an OpenStack-native and CloudFormation-compatible API. It is partially intended to facilitate movement of workloads from AWS to OpenStack deployments. Heat was promoted from incubation status to an integrated component of OpenStack in the Grizzly release (April 2013).

NetApp Storage Solutions

This section describes NetApp storage solutions that leverage NetApp clustered Data ONTAP® and E-Series and EF-Series storage systems with OpenStack. It also discusses highly available enterprise-class storage for OpenStack.

Leveraging NetApp Clustered Data ONTAP with OpenStack

NetApp clustered Data ONTAP helps to achieve results and deliver products to market more quickly by providing the throughput and scalability needed to meet the demanding requirements of high-performance computing and digital media content applications. It also facilitates high levels of performance, manageability, and reliability for large virtualization and cloud platforms.

Clustered Data ONTAP provides the following advantages:

- Scale-up, scale-out, and scale-down are possible with numerous nodes by using a global namespace.
- Storage virtualization with storage virtual machines (SVMs, formerly called Vservers) eliminates the physical boundaries of a single controller (memory, CPU, ports, disks, and so forth).
- Nondisruptive operations (NDO) are available when you redistribute load or rebalance capacity, in addition to network load-balancing options in the cluster for upgrading or expanding its nodes.
- NetApp storage efficiency features (such as NetApp Snapshot® copies, deduplication, data compression, and NetApp RAID DP® thin provisioning) and space-efficient cloning technology are also available.

Some of the problems solved by clustered Data ONTAP include:

- **Scalable capacity.** Grow capacity incrementally, on demand, through the nondisruptive addition of storage shelves and the growth of storage containers (pools, LUNs, file systems). Support nondisruptive redistribution of existing data to the newly provisioned capacity as needed through volume moves.
- **Scalable performance and pay as you grow.** Grow performance incrementally, on demand and nondisruptively, through the addition of storage controllers in small, economical (pay-as-you-grow) units.
- **High availability (HA).** Leverage highly available pairs to provide continuous data availability in the face of individual component faults.
- **Flexible, manageable performance.** Support different levels of service and provide the ability to dynamically modify the service characteristics associated with stored data by nondisruptively migrating data to slower, less costly disks and/or by applying quality of service (QoS) criteria.

- **Scalable storage efficiency.** Control costs through the use of scale-out architectures that employ commodity components. Grow capacity and performance on an as-needed (pay-as-you-go) basis. Increase utilization through thin provisioning and data deduplication.
- **Unified management.** Provide a single point of management across the cluster. Leverage policy-based management to streamline configuration, provisioning, replication, and backup. Provide a flexible monitoring and reporting structure for implementing an exception-based management model. Virtualize resources across numerous controllers so that volumes become simple-to-manage logical entities that span storage controllers for performance and dynamic redistribution of data. Place customer and application data on a proven and stable enterprise platform.

Leveraging NetApp E-Series and EF-Series Storage Systems with OpenStack

The family of NetApp E-Series and EF-Series systems provides performance-efficient, high-density block storage aimed primarily at application-driven workloads.

NetApp E-Series and EF-Series systems are designed to provide:

- Simple, low-touch administration
- Flexible, reliable SAN storage
- Extreme capacity and density
- High-performance GB/sec or IOPS
- Performance, power, and space efficiency
- Consistent low latency
- Consistently high IOPS and throughput
- Enterprise-class capabilities (reliability, availability, manageability)

The NetApp EF-Series is an all-flash storage array that brings together extreme performance and enterprise-grade reliability to create a system optimized for latency-sensitive workloads. Building on a heritage of performance leadership, its core architecture has been proven in some of the world's most demanding and complex computing environments. Its field-proven design is the culmination of 20 years of industry knowledge focused on designing enterprise-class storage. Leveraging experience from over 650,000 systems shipped, the fully redundant EF-Series all-flash array is architected to provide high levels of reliability, availability, and data protection.

Both the E-Series and the EF-Series systems run on the enterprise-proven NetApp SANtricity® software platform. SANtricity is designed to combine sustainable low-latency performance with enterprise-class availability and protection. Its streamlined firmware is well suited to the extreme demands of latency-sensitive workloads. SANtricity helps keep I/O flowing with automated path failover, online administration, advanced data protection, proactive monitoring and repair, nondisruptive upgrades, and extensive diagnostic capabilities.

Highly Available Enterprise-Class Storage for OpenStack

With capabilities such as self-healing and integrated data protection for backup and disaster recovery, NetApp solutions are enterprise proven and help reduce risk for OpenStack cloud deployments. A global network of service provider partners has already deployed hundreds of high-service-level-agreement (SLA) cloud services built on NetApp storage with over a billion users worldwide.

Because NetApp technology is integrated with OpenStack Block Storage Service, OpenStack Object Storage Service, OpenStack Image Service, and OpenStack Compute Service, users can build on this proven and highly scalable storage platform to optimize their private and public cloud architectures by reducing risk and increasing efficiency.

NetApp's OpenStack integrations expose unique and differentiated capabilities:

- **Cinder with NetApp FlexClone® technology.** You can leverage NetApp FlexClone technology to create fast and efficient Cinder volume snapshots, as well as use it when Cinder snapshots serve as the source content for new Cinder volumes.
- **Glance and deduplication.** You can leverage the deduplication technology of Data ONTAP for NetApp FlexVol® volumes that provide storage for Glance because this creates a storage-efficient Glance image store.
- **Nova and enhanced instance creation.** NetApp FlexClone technology enables the instant provisioning of new Cinder volumes that serve as the persistent root disk for VMs rather than the default behavior of redownloading images from Glance for each new VM provisioned.
- **Protocol options.** Through the unified architecture of Data ONTAP, any version of NFS (NFS v3, v4.0, v4.1, or pNFS) or iSCSI can be leveraged in OpenStack deployments. This removes the need to deploy multiple storage products in order to address protocol requirements.
- **Swift with E-Series.** The E-Series Dynamic Disk Pools (DDP) feature helps reduce the storage footprint and enhance protection and scalability. This helps to reduce network traffic for replicas of Swift objects because of E-Series storage efficiencies.
- **Scalability.** NetApp clustered Data ONTAP allows cloud administrators to take advantage of the ability to move NetApp FlexVol volumes between nodes of a cluster to balance, distribute, and improve performance of the cloud infrastructure without interrupting service.
- **Data protection.** The nondisruptive operational capabilities of Data ONTAP help storage operations continue to succeed without loss of service in the event of lost disks.

Important

With the advantage of NetApp clustered Data ONTAP, an OpenStack deployment can scale on demand without losing flexibility, security, performance, or manageability. VMs can be deployed in a highly accelerated manner from the storage array. Applications can operate within secure tenants that span the entire hardware, network, and storage stack. Major components of the storage infrastructure can be replaced or upgraded nondisruptively.

By allowing maximum operations of OpenStack-based infrastructures, NetApp further increases return on investment and total cost of ownership for the entire environment. NetApp uses the following technologies to provide a resilient OpenStack infrastructure:

- **Clustered NetApp controllers.** Clustered controllers deliver enterprise-class availability by providing controller redundancy and simple, automatic, transparent failover in the event of a storage controller failure. Providing transparent recovery from component failure is critical because VMs rely on the shared storage. For more information, refer to NetApp TR-3450: High-Availability Pair Controller Configuration Overview and Best Practices. Clustered controllers use pairs of storage controllers, which allow each pair of nodes to serve data simultaneously and still have failover capability with each other.
- **Multipath HA.** The multipath HA storage configuration improves the resiliency and performance of active-active controller configurations. Multipath HA storage configurations enhance storage resiliency by reducing unnecessary takeover by a partner node because of a storage fault, which improves overall system availability and promotes higher performance consistency. Multipath HA provides added protection against various storage faults, including host bus adapter or port failure, controller-to-shelf cable failure, shelf module failure, dual intershelf cable failure, and secondary path failure. Multipath HA helps achieve consistent performance in active-active configurations by providing larger aggregate storage loop bandwidth. For more information, refer to NetApp TR-3437: Storage Subsystem Resiliency Guide.
- **RAID data protection.** Protecting data against disk drive failure by using RAID is a standard feature of most shared storage devices. However, with the capacity and subsequent rebuild times of current hard drives, for which exposure to another drive failure can be catastrophic, protection against double disk failure is now essential. NetApp RAID DP, an advanced RAID technology, is provided as the default RAID level on all FAS systems. RAID DP provides performance comparable to that of RAID

10, with much higher resiliency. It provides protection against double disk failure as compared to RAID 5, which can protect only against a single disk failure in a RAID group. NetApp strongly recommends using RAID DP on all RAID groups that store VMware View data. For more information about RAID DP, refer to NetApp TR-3298: RAID-DP: NetApp Implementation of Double-Parity RAID for Data Protection.

- **Dynamic Disk Pools (DDP).** The E-Series DDP feature provides an alternative to standard RAID groups. DDPs simplify protection by removing the complexity of configuring RAID groups and allocating hot spares. Utilization is improved by dynamically spreading data, parity, and spare capacity across all drives in a pool, reducing performance bottlenecks caused by hot spots. In addition, if a drive failure occurs, DDP enables the return to an optimal state significantly faster than RAID 6, while reducing the performance impact during the reconstruction of a failed drive. DDP also offers greater protection from multiple drive failures by prioritizing the reconstruction of the most critical segments.

1.3 Use Case Summary

An OpenStack deployment is capable of supporting a variety of applications and workloads in a fully multi-tenant environment. Its set of open, consistent APIs provides an abstraction to the underlying hypervisors, OSs, network devices, and storage solutions deployed. OpenStack can be used to provide a self-service IaaS on which the following use cases can be deployed:

- Development and test workloads
- Shared services deployed in support of mobile applications
- Highly elastic web workloads
- Disaster recovery scenarios
- Primary workloads
- Cloudburst for peak workloads

Note: The OpenStack solution architecture described in this document is not limited to the use cases listed.

2 Design Fundamentals

It is extremely common for end users of infrastructure services to expect that the services have HA. Although continuous availability (100% uptime) can be achieved in certain circumstances, it can become prohibitively expensive to provide this standard of service. There is a fundamental tradeoff between how available a service can be and the cost, architectural complexity, and operational processes required to deliver that availability. The service-level objective (SLO), typically expressed as a percentage of time (for example, 99.99%), indicates the amount of time that the end user of a service can expect the service to respond successfully to basic functional requests. Higher levels of availability are typically achieved by deploying systems with increasing levels of redundancy and fault tolerance.

To deliver a service with high SLOs, the key design tenets should include:

- Removing single points of failure
- Minimizing component downtime
- Minimizing the possibility of data loss

2.1 Deployment Roles

Because an OpenStack deployment is typically composed of several of the services described in section 1, “Solution Overview,” it is important to logically separate the services that provide management of underlying infrastructure resources from those that provide access to resources provisioned by tenants. Describing the functional responsibilities of nodes in an OpenStack deployment through the concept of

roles makes it easier to maintain this separation, which allows a more straightforward deployment topology that appropriately addresses concerns related to both control and data.

Four primary roles are part of this solution: the controller node, the compute node, the network node, and the object storage node.

Role of Controller Node

The controller role represents the nodes that provide the control plane for an OpenStack deployment. Controller nodes expose API endpoints that are consumed by external tenants of the cloud, as well as other OpenStack services in a deployment. In the case of most OpenStack services, a single process exposes the API endpoint for that service. In addition, other auxiliary processes that typically coreside with the API endpoint communicate with other processes in the same service over a distributed messaging bus and store state in a SQL database (see section 4, “Database and Messaging Subsystems”).

Controller nodes should reside on servers that are sized appropriately for handling potentially high volumes of RESTful transactions (that is, optimized for CPU and network traffic).

Role of Compute Node

The compute role represents the nodes that provide the data plane for an OpenStack deployment, namely, the actual computing resources to external tenants of the cloud. In most cases, computing resources are VMs that run in the context of a hypervisor. The compute nodes (hypervisor hosts) communicate with the other OpenStack services over the distributed messaging bus and/or through REST APIs.

Kernel-based Virtual Machine (KVM) is commonly used as the hypervisor in OpenStack deployments; therefore, the Linux system that runs KVM operates under the role of a compute node in the deployment topology. The same is true for the Microsoft Hyper-V and LXC virtualization platforms. If VMware vCenter is leveraged as the hypervisor in an OpenStack deployment, the OpenStack computing services are likely to reside on the controller node that communicates with VMware vCenter through web services calls. When Xen or VMware ESXi is leveraged, the compute services are hosted on a guest VM running on the hypervisor host that is being managed.

Compute nodes should reside on servers that are sized appropriately for hosting virtualization workloads.

Role of Network Node

The network role represents the nodes that provide the network fabric for an OpenStack deployment when open-source software-defined network components are used instead of dedicated, specialized network routers and switches. The network nodes communicate with the other OpenStack services over the distributed messaging bus and/or through REST APIs. If you expect that instances will generate significant traffic either to or from the Internet, a dedicated network node helps avoid CPU contention between the Neutron L3 agent and other OpenStack services that forward packets.

In the Juno release, Neutron added the distributed virtual router (DVR) capability, which allows deployers to deploy the L3 routers concurrently with compute nodes rather than requiring a dedicated network node. This capability allows network traffic between VMs on the same compute node to be processed locally without having to traverse the network node. VMs that have externally addressable floating IP addresses can also directly forward traffic to external networks with the DVR function enabled. DVR removes some significant scaling issues and also helps to address architectural issues that imposed significant requirements on highly available Neutron deployments.

Network nodes should reside on servers that are sized appropriately for network-centric workloads.

Role of Object Storage Node

Most OpenStack services make available a distributed control plane for managing infrastructure resources delivered to end users of a cloud. One exception to this model is the OpenStack Object Storage (Swift) service, which provides both control and data plane functionality for the service. In addition, object storage services tend to be optimized for customer environments that have significant demands on capacity as well as scalability.

For these reasons, there is a separate role for nodes hosting object storage services in an OpenStack deployment, referred to in this solution design as an object storage node. Object storage nodes are referred to as proxy nodes when they host the API endpoint of Swift or as storage nodes when they manage the storage and replication of account, container, and object data.

Like most other OpenStack services, Swift has separate processes that expose the API endpoint and manage the underlying storage resources, replication policies, and so forth. Object storage nodes should reside on servers that are sized appropriately for handling large amounts of network traffic (for client requests, replication, and connectivity to the storage backend over the iSCSI protocol).

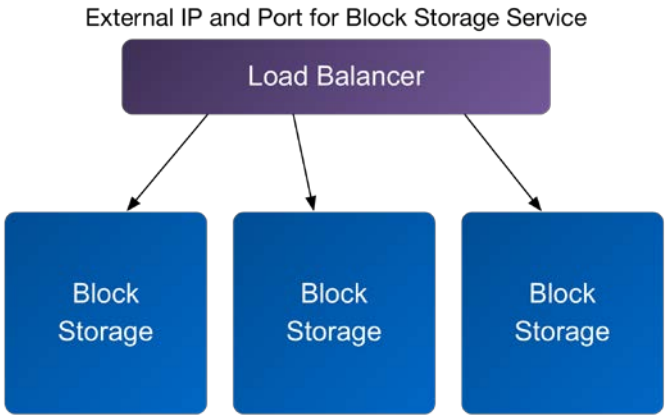
2.2 Active-Passive Versus Active-Active Topologies

Stateless services do not maintain any information (state) related to a particular transaction, so subsequent transactions depend on being processed by the same entity that processed a previous request. In contrast, stateful services do maintain state, so it is important that transactions can access previously stored information in an entity in support of processing the current request.

Active-Active Topology

The vast majority of OpenStack services are stateless in nature and communicate between other processes within the scope of the same service over distributed message queues. Therefore, the stateless API endpoints can be replicated across multiple nodes and placed behind a load balancer (as shown in Figure 2) to provide resiliency to failure and increased performance.

Figure 2) Active-active deployment logical topology.



Both the underpinning database and the messaging subsystem components of OpenStack deployments can be deployed in active-active topologies. For more detail on databases and messaging subsystems deployed in an active-active topology, see section 4, “Database and Messaging Subsystems.”

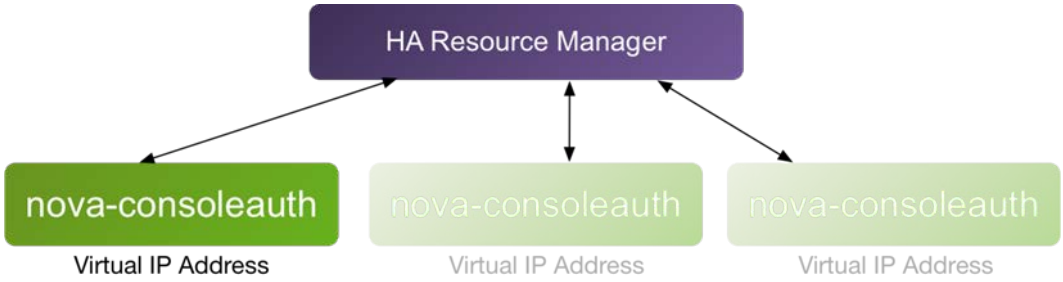
Active-Passive Topology

One of the processes in the OpenStack services described in this solution is stateful and cannot have multiple deployments processing requests concurrently. The Nova token console authorization process

(`nova-consoleauth`) must be run in an active-passive topology in an OpenStack deployment. An HA resource manager (such as Pacemaker) is required to monitor the status of a process that can be operational on only one node in a deployment and to start it on another node if the resource manager determines that it is no longer running. The resource manager also enforces having only one copy of a managed service operational at a time in the deployment. Figure 3 shows the logical topology of an active-passive deployment.

Note: In the Icehouse release of OpenStack, the Neutron L3 agent (neutron-l3-agent when Neutron is deployed without Provider Network extensions) was required to be deployed in an active-passive topology. With the introduction of Virtual Router Redundancy Protocol (VRRP) support and the DVR capability in the Juno release, the active-passive topology is no longer required.

Figure 3) Active-passive deployment logical topology.



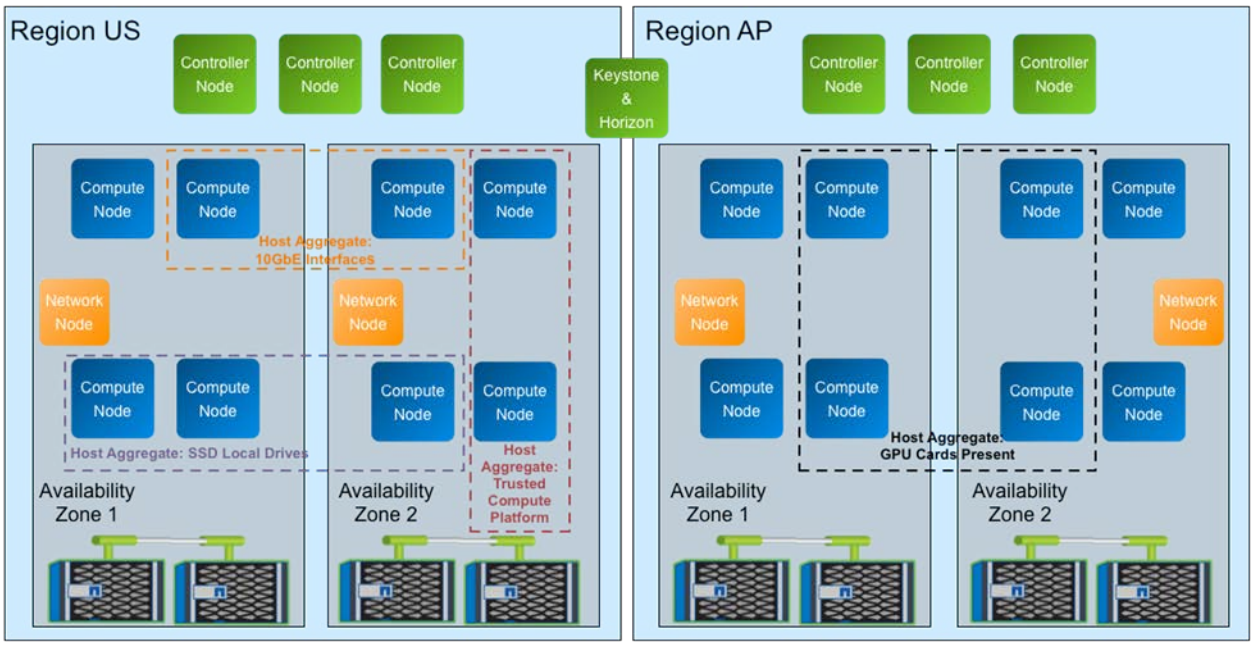
Both the underpinning database and the messaging subsystem components of OpenStack deployments can be deployed in active-passive topologies. For more details about databases and messaging subsystems deployed in an active-passive topology, see section 4, “Database and Messaging Subsystems.”

2.3 Regions, Cells, Availability Zones, and Host Aggregates

For cloud administrators who must design deployments involving physical or logical separation of resources in order to address end users’ functional requirements (such as HA or resource scheduling) or nonfunctional requirements (such as legal concerns about physical location of data), OpenStack offers several useful concepts. Regions, cells, availability zones, and host aggregates all present different segmentation mechanisms that allow cloud end users to place workloads on an OpenStack deployment to achieve resiliency against a variety of failure domains.

Figure 4 shows three of these OpenStack concepts: regions, availability zones, and host aggregates.

Figure 4) Logical diagram: regions, availability zones, and host aggregates.



Regions

Regions represent separate, distinct deployments of OpenStack services that have completely separate API endpoints. All OpenStack services except for Horizon and Keystone are replicated on a per-region basis. Typically, regions represent physically separate but logically related OpenStack deployments (for example, different cities, countries, or continents).

Figure 4 shows two regions, US and AP, reflecting a deployment in the United States and another in the Asia-Pacific region.

Availability Zones

Availability zones represent logical separations of a region (that is, an OpenStack deployment) based on some specified criterion. For example, different floors of a data center building (with separate power, network, and cooling infrastructure) might each reflect an availability zone based on how the underlying infrastructure was segmented. Workloads can be deployed across different availability zones to achieve greater resiliency to failure without the concerns associated with traversing a wide area network.

Figure 4 shows two availability zones in each region, each with its own compute nodes and storage infrastructure but with shared OpenStack services running on the controller nodes.

Host Aggregates

Like availability zones, host aggregates provide a logical separation of resources in a region; however, unlike availability zones, which are visible to end users, host aggregates are visible only to cloud administrators. Host aggregates provide a link between a set of capabilities that a compute node can provide and the definition of a flavor that maps to a selected subset of capabilities. End users select flavors when they request that new instances be provisioned.

Host aggregates are flexible in that combinations of arbitrary key/value pairs can be assigned to represent the meaning of a host aggregate and the same key/value pair can be assigned to multiple host aggregates. A compute node can be a member of multiple host aggregates, but it can be in only a single availability zone and a single region. All members of a host aggregate must be in the same region.

Figure 4 shows three different host aggregates defined in the US region and one host aggregate defined in the AP region. The definition of these host aggregates typically corresponds to some functional characteristics of compute nodes. The characteristic might be, for example, that some compute nodes have SSD local disks that provide better performance for ephemeral storage or that others run on top of hardware that has attested that it runs on a trusted computing platform.

Note: The concept of host aggregates is leveraged only in the OpenStack Compute Service (Nova), whereas the concept of availability zones is supported in both Nova and Cinder. Some resources, including images, floating IP addresses, networks, security groups, and key pairs, have regionwide scope.

Cells

The cell deployment model provides an alternative to deploying by regions. Cells represent a logical segmentation of compute resources exposed through a single API endpoint. Through the use of a hierarchical scheduler along with segmented database and queueing subsystems, cells allow compute resources to be deployed independently and yet to be abstracted through a single interface. With the introduction of some additional mapping capabilities, the Cells V2 implementation that was added during the Kilo release is now considered stable for wide-scale deployments; in fact, the default installation of Nova is now a single-cell deployment.

Application Topology Design

When considering how to leverage these concepts from an end user's perspective, it is important to consider questions such as these:

- How isolated are resources in different availability zones in the same region?
- Is my application extremely sensitive to potential latency for interregion communication?
- Are there legal or political reasons to locate an application in a particular region?

Best Practice

NetApp strongly recommends replicating appropriate components of an application topology across cells or different availability zones and different regions according to the SLO of the application and the resiliency-to-failure domains provided by the concepts related to a particular OpenStack deployment.

2.4 Deploying and Using TripleO: OpenStack on OpenStack

TripleO (OpenStack on OpenStack, also known as OpenStack Management) is an OpenStack incubated project that aims to facilitate planning, deployment, and ongoing operations of OpenStack deployments. It provides constructs that allow cloud administrators to:

- Programmatically declare the topology of the desired deployment, including the types and capacities of resources available from which to build the cloud.
- Deploy OpenStack according to the declared topology, including bare-metal installation of host OSs, hypervisors, and OpenStack services, through a policy-driven process.
- Operate the OpenStack deployment by visualizing capacity and other metrics through an intuitive user interface.

TripleO delivers this capability by combining existing OpenStack projects, such as Heat, Neutron, Glance, Ceilometer, and Nova, with other initiatives, such as Ironic and Tuskar, to create at least two clouds:

- **The overcloud**, which is the cloud for end users and what most people refer to when they mention OpenStack

- **The undercloud**, which is the command and control cloud responsible for managing the physical resources available to an OpenStack deployment and assembling them so that one or more overclouds can be provisioned on it

At the time of this writing, some OpenStack distributions (for example, Red Hat Enterprise Linux OpenStack Platform) either are working on or have released distribution installers based on the TripleO framework; however, the project scope of TripleO includes more than just installing OpenStack. It is meant to provide a holistic management capability specifically targeted to administrators of OpenStack deployments who are responsible for the entire lifecycle of a cloud.

3 Networking Best Practices

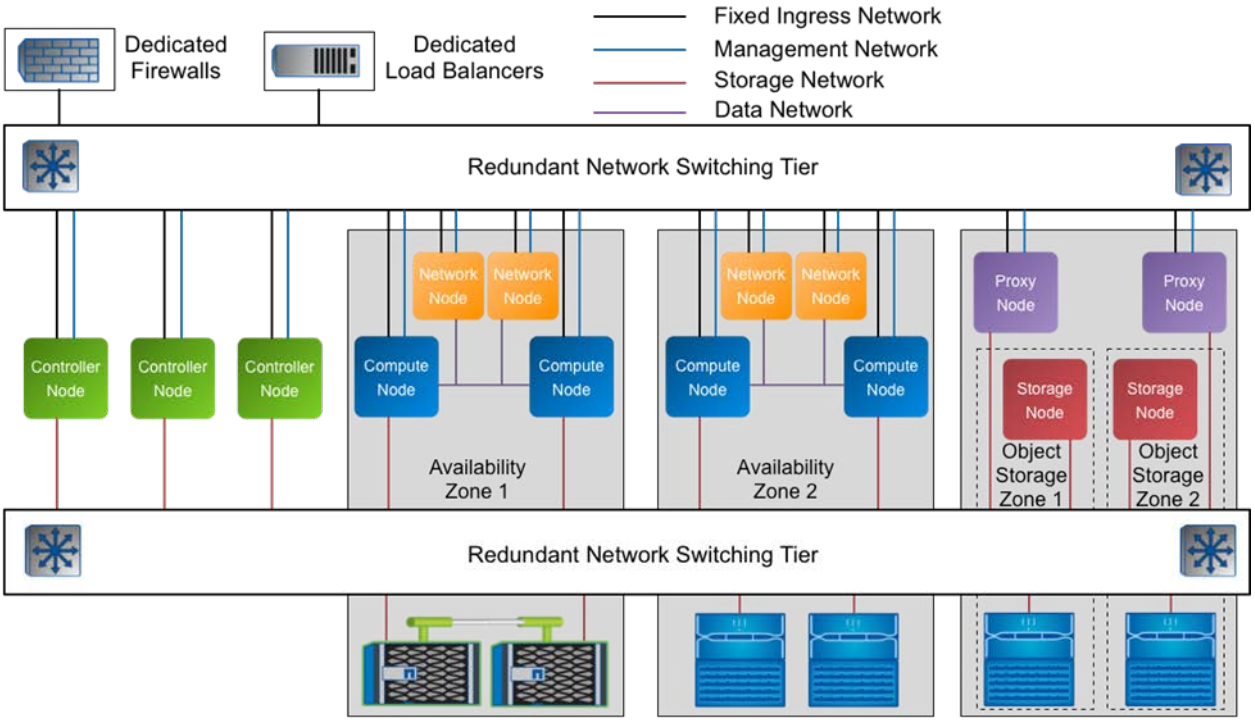
The goal of any network is to provide uninterrupted service to all nodes that connect to it. This section focuses primarily on how to establish a highly available network on which to deploy OpenStack.

Regardless of protocol, a network that underpins an OpenStack deployment must address these goals:

- Be redundant across switches in a multiswitch environment.
- Use as many available physical paths as possible.
- Be scalable across multiple physical interfaces or ports.

Figure 5 shows an example of a high-level networking layout for a single region of an OpenStack deployment. All network links in Figure 5 reflect bonded interfaces, and the different networks can be segmented physically or logically through the use of virtual local area networks (VLANs). Note that there are redundant controller nodes for the entire region, as well as compute and storage resources that are grouped into different availability zones. For the object storage deployment, there are two zones that each contain proxy and storage nodes. Although only one proxy node and one storage node are shown, a normal deployment would have multiple nodes of each type in a zone. For more information about Swift deployments, see section 9, “OpenStack Object Storage Service (Swift).”

Figure 5) High-level networking diagram.



3.1 API Endpoints

OpenStack services typically expose three separate API endpoints (URLs): public, internal, and administrative. These endpoints are described in detail in Table 1.

Table 1) API endpoints for OpenStack services.

Endpoint	Purpose
Public URL	End users typically connect through this URL to interact programmatically with OpenStack services.
Internal URL	End users (within an internal network) can interact programmatically with OpenStack services through this URL. In some public cloud environments, it might be desirable to connect to OpenStack services from instances running in an OpenStack deployment without connecting through a public IP, avoiding firewalls or data usage charges by using an egress interface.
Administrative URL	Administrative activities on each of the OpenStack services are conducted through this URL. It typically listens on a different port from that of the public or internal URLs (to avoid confusion) and potentially even uses a different uniform resource identifier (URI).

Note: If OpenStack services are deployed in an active-active manner (as described in Section 2.2), it is important to use the host name or IP address of the load balancer when constructing these URLs and subsequently registering them as endpoints with Keystone.

3.2 Management Networks

In considering the logical topology for controller and compute nodes, it is important to consider the networks that are required to support the management and control plane. It is also important to consider the networks that will carry end-user traffic (both ingress and egress from compute instances as well as the storage traffic between controller and compute nodes and the backend storage subsystem).

This section divides Figure 5 into several logical views showing the relationship between controller nodes, compute nodes, and NetApp storage systems with regard to management network traffic.

Figure 6 shows the different management networks that exist when a NetApp Data ONTAP cluster is deployed in a larger OpenStack deployment. The Data ONTAP cluster has its own interconnect network that is used for a variety of purposes, including health checks, failover support, and intracluster data traffic. The storage management network connects the controller nodes to the Data ONTAP cluster and carries provisioning requests in addition to metering and monitoring traffic.

Figure 6) Logical topology: NetApp Data ONTAP and OpenStack management networks.

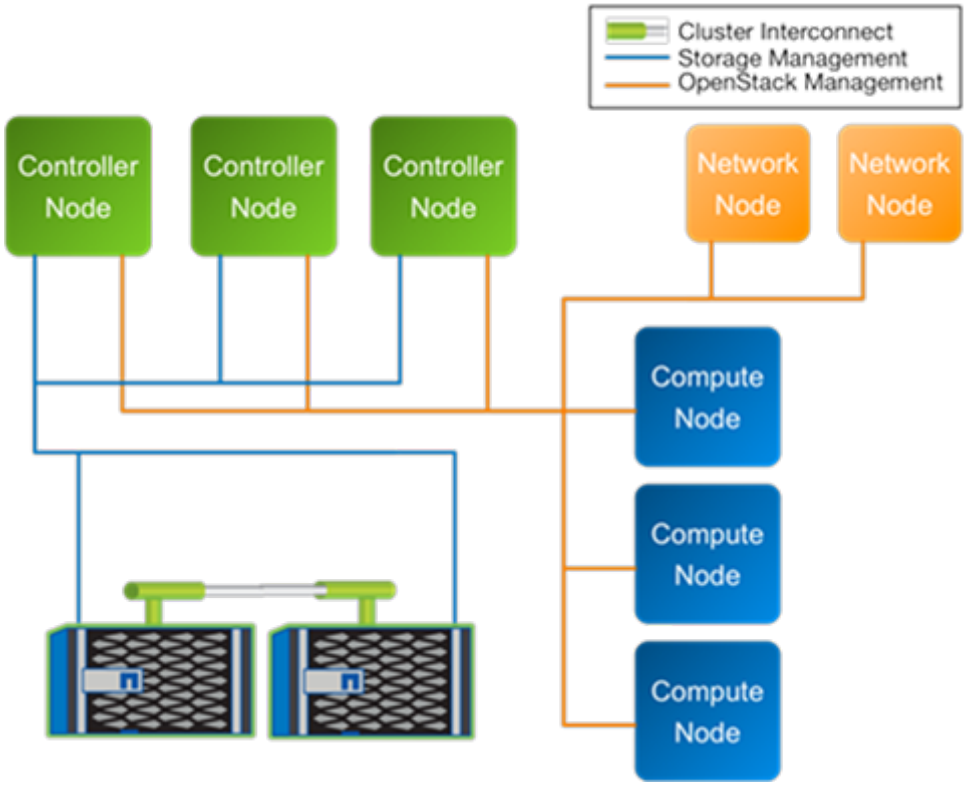
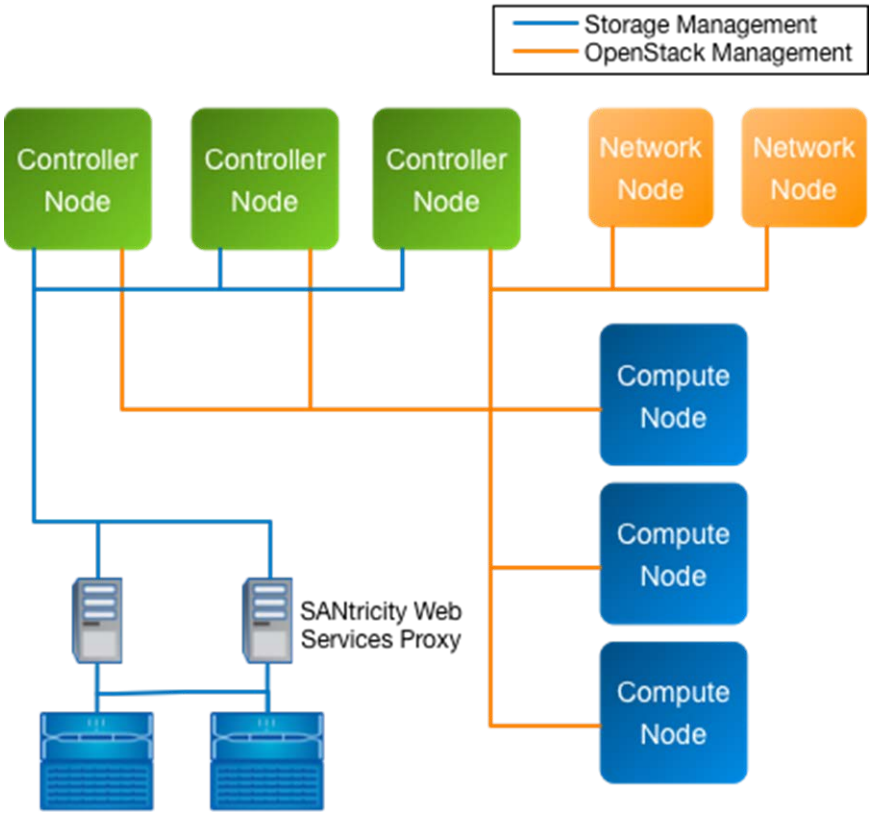


Figure 7 shows the different management networks that exist when a NetApp E-Series storage controller is deployed in an OpenStack deployment. The storage management network connects the controller nodes to SANtricity Web Services Proxy and the E-Series storage system to the proxy node. This network carries storage provisioning requests as well as metering and monitoring traffic in the OpenStack deployment.

Figure 7) Logical topology: NetApp E-Series and OpenStack management networks.



The OpenStack management network shown Figure 6 and Figure 7 handles the various types of interprocess communication traffic generated by the OpenStack services in addition to the database and messaging services that underpin the various OpenStack services.

3.3 Storage Networks and Fabrics

The following sections discuss storage networks and explain the use of Fibre Channel (FC) as a storage protocol for deployments based on Data ONTAP.

Storage Networks

This section divides Figure 5 into several logical views that show the relationship between controller nodes, compute nodes, and NetApp storage systems with regard to storage network traffic.

The public network and the tenant overlay networks are shown in both Figure 8 and Figure 9. End users of an OpenStack deployment connect to the public API endpoints over the public network in addition to accessing the VM instances they provisioned in the cloud. Tenant overlay networks allow end users to create their own networking topologies that segment resources within the cloud.

Figure 8 shows the different storage data networks that exist when a NetApp Data ONTAP cluster is deployed within a larger OpenStack deployment. The storage data network connects the compute nodes with the storage subsystems that store block storage volumes, VM images, and ephemeral instance storage.

Note: The storage data network is also connected to the controller nodes hosting the OpenStack Block Storage (Cinder) service because the controller nodes mount the NFS exports or attach to iSCSI LUNs to perform provisioning operations and metering and monitoring activity.

Figure 8) Logical topology: NetApp Data ONTAP and OpenStack storage networks.

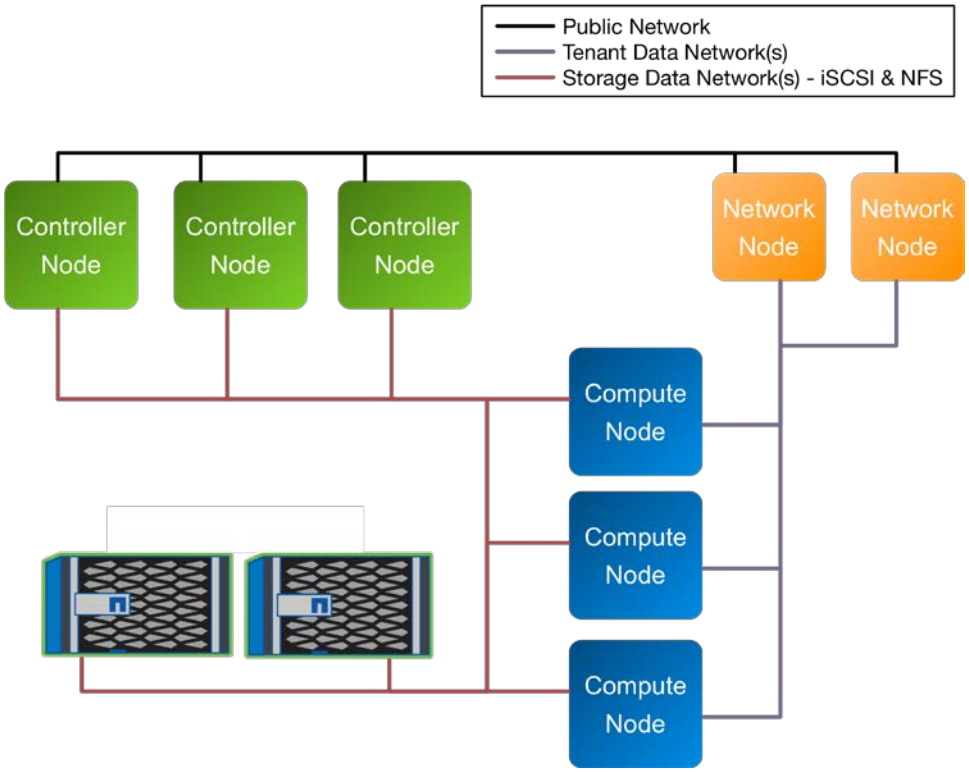
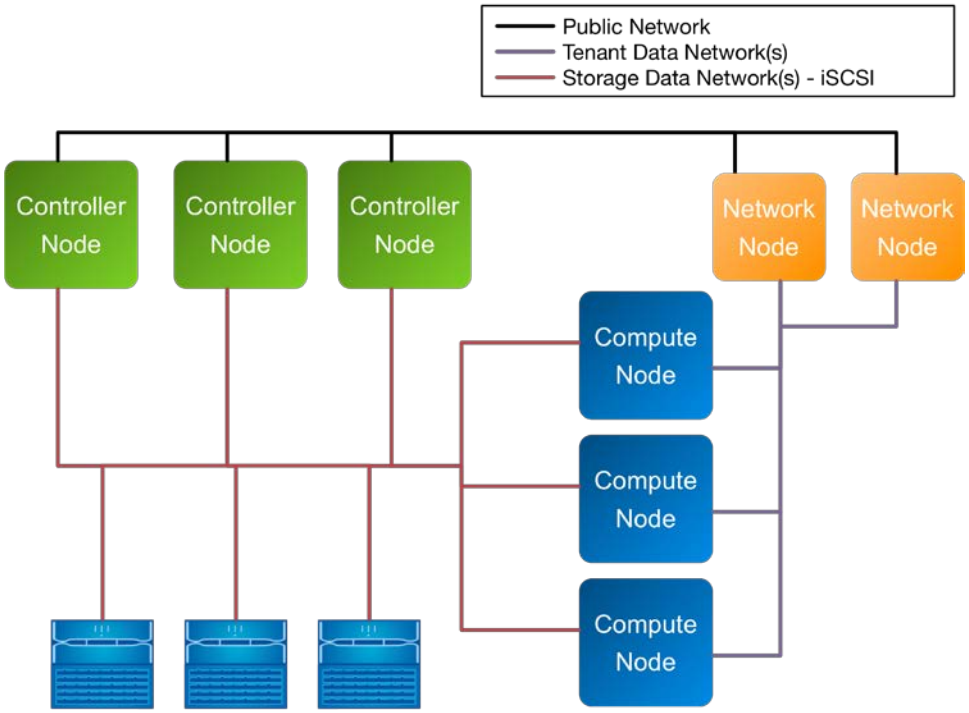


Figure 9 shows the different storage data networks that exist when NetApp E-Series storage is deployed within a larger OpenStack deployment. The storage data network connects the compute nodes with the storage subsystems that store block storage volumes. The storage data network is also connected to the controller nodes because the controller services might have to attach a LUN to download an image from Glance and write its contents into an appropriate destination LUN.

Figure 9) Logical topology: NetApp E-Series and OpenStack storage networks.



Best Practices

NetApp recommends the following resiliency best practices when deploying storage networks:

- Use RAID DP, the NetApp high-performance implementation of RAID 6, for better data protection with Data ONTAP.
- Use Dynamic Disk Pools to achieve a faster and more efficient parity protection scheme with E-Series and EF-Series storage solutions.
- Use multipath HA with active-active storage configurations to improve overall system availability and promote higher performance consistency.
- Allow Data ONTAP to select disks automatically when creating aggregates or volumes.
- Use the latest Data ONTAP or SANtricity general deployment release available on the NetApp Support site.
- Use the latest storage controller, shelf, and disk firmware available on the NetApp Support site.
- Use clustered storage controllers to eliminate single points of failure.

For more information, refer to NetApp TR-3437: Storage Subsystem Resiliency Guide.

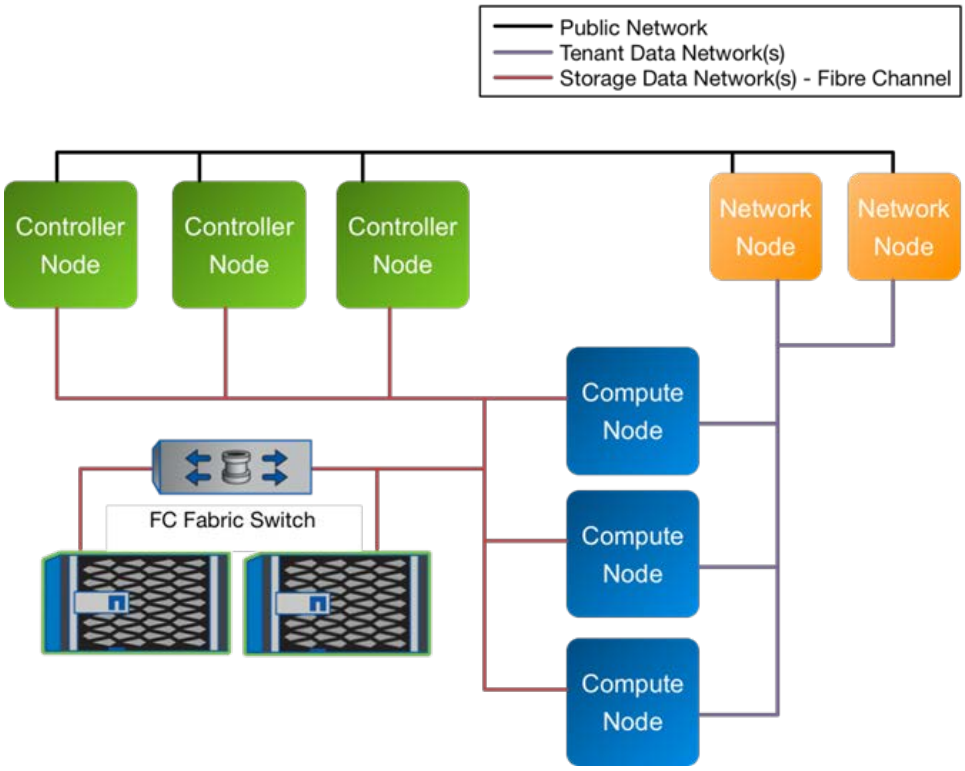
Fibre Channel Fabrics

In the Kilo release of OpenStack, NetApp added support for leveraging FC as a storage protocol for deployments based on Data ONTAP.

Figure 10 shows the different storage data networks that exist when a NetApp Data ONTAP cluster is deployed in a larger OpenStack deployment. The storage data network connects the compute nodes with the storage subsystems that store block storage volumes, VM images, and ephemeral instance storage.

Note: The FC fabric is also connected to the controller nodes hosting the OpenStack Block Storage (Cinder) service because the controller nodes attach to LUNs to perform provisioning operations and metering/monitoring activity.

Figure 10) Logical topology: NetApp Data ONTAP and OpenStack FC fabrics.



Fibre Channel Zoning

OpenStack controller nodes and NetApp storage arrays can connect to a SAN fabric by using host bus adapters (HBAs). Each HBA can run as either an initiator (OpenStack) or a target (NetApp). Each adapter has a global unique address referred to as a World Wide Name (WWN). Each WWN must be known in order to configure LUN access on a NetApp storage array.

Many devices and nodes can be attached to a SAN. Implementing zones is a method used to secure access and optimize I/O access to these devices. SAN zoning is a method of arranging FC devices into logical groups over the physical configuration of the fabric or FC network.

Zoning is available in hardware (hard zoning) or in software (soft zoning). An option available with both implementations is port zoning. With this option, physical ports define security zones. A host's access to a LUN is determined by the physical port to which it connects. With port zoning, zone information must be updated every time a user changes switch ports. In addition, port zoning does not allow zones to overlap.

Another form of zoning is World Wide Port Name (WWPN) zoning, in which the fabric leverages its name servers to either allow or block access to particular WWPNs in the fabric. A major advantage of WWPN zoning is the ability to recable the fabric without having to modify the zone members.

The NetApp FC driver for OpenStack uses the WWPN zoning method for both clustered Data ONTAP and Data ONTAP operating in 7-Mode.

Best Practices

NetApp recommends the following resiliency best practices when deploying FC fabrics:

- Have at least two adapter ports in each OpenStack controller node running Cinder services with the FC protocol.
- Implement “single-initiator, multiple-storage target” zones.
- Use multipath HA with active-active storage configurations to improve overall system availability and promote higher performance consistency.
- Follow NetApp guidelines in defining the Host MPIO/ALUA configuration as recommended in the NetApp Host Utilities Kit documentation.
- Allow Data ONTAP to select disks automatically when creating aggregates or volumes.
- Confirm that the FC and host infrastructure configuration is supported in the NetApp Interoperability Matrix Tool (IMT).
- Use the latest Data ONTAP general deployment release available on the NetApp Support site.
- Use the latest storage controller, shelf, and disk firmware available on the NetApp Support site.
- Use clustered storage controllers to eliminate single points of failure.

For more information, refer to NetApp TR-3437: Storage Subsystem Resiliency Guide and to NetApp TR-4080: Best Practices for Scalable SAN in Clustered Data ONTAP 8.2.

3.4 Object Storage Networks

This section divides Figure 5 into several logical views showing the relationship among proxy nodes, storage nodes, and NetApp storage systems in an OpenStack Object Storage deployment.

The public network and the Swift lookup and replication network are shown in both Figure 11 and Figure 12. End users of an OpenStack deployment connect to the public Swift API endpoint over the public network. The Swift lookup and replication network connects the proxy nodes to the storage nodes and carries all traffic related to establishing and maintaining the ring. For more information about Swift, see section 9, “OpenStack Object Storage Service (Swift).”

Figure 11 shows the networks that exist when a Swift is deployed atop a NetApp Data ONTAP cluster. The Data ONTAP cluster has its own interconnect network that is used for a variety of purposes, including but not exclusive to health checks, failover support, and intracluster data traffic. The storage data network connects the storage nodes with the storage subsystems containing LUNs used by Swift to store account, container, and object data.

Figure 11) Logical topology: NetApp Data ONTAP and OpenStack object storage networks.

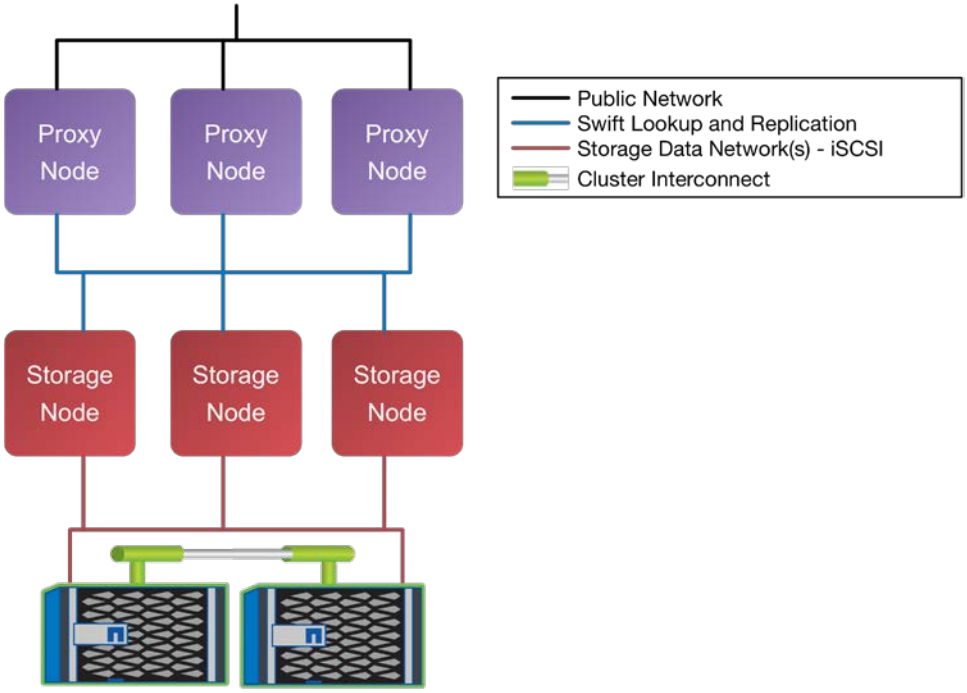
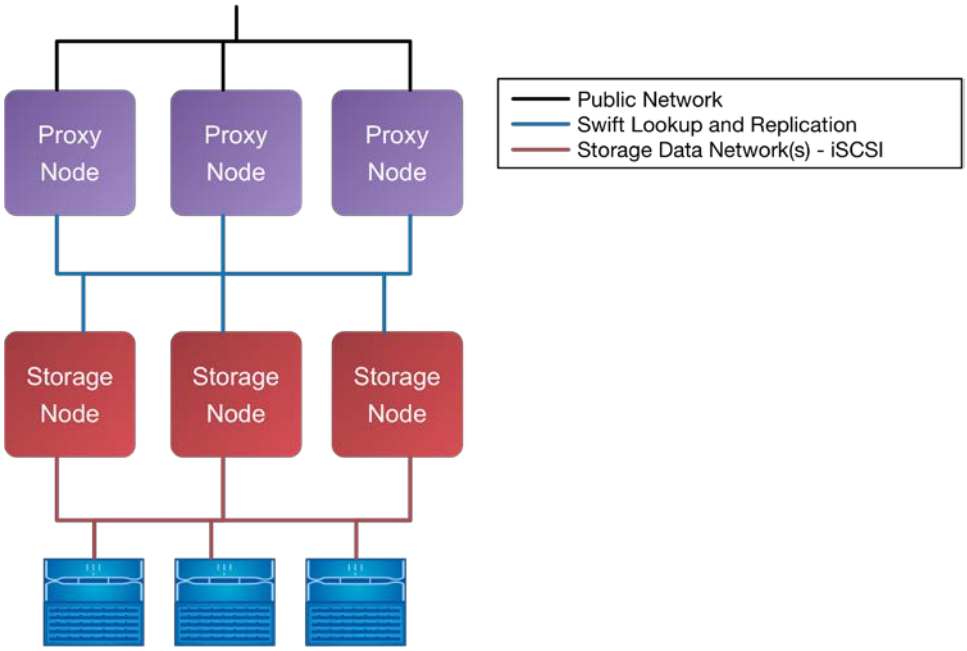


Figure 12 shows the different networks that exist when Swift is deployed atop NetApp E-Series storage. The storage data network connects the storage nodes with the storage subsystems containing LUNs used by Swift to store account, container, and object data.

Figure 12) Logical topology: NetApp E-Series and OpenStack object storage networks.



Best Practice

NetApp E-Series and EF-Series storage systems support connectivity to the storage controller through SAS, iSCSI, Infiniband, and FC. The choice for the interconnect should match the higher-level requirements for elasticity and performance for the object storage infrastructure. In making the decision, consider the following guidelines:

- iSCSI connectivity over 10GbE might provide a more elastic environment that could ease scale-out of an object storage deployment.
- SAS, Infiniband, or FC might provide superior performance.
- SAS has the added advantage of not requiring a switch for interconnectivity and thus provides a superior mix of performance and cost effectiveness.

3.5 Networking Best Practices

For proper network redundancy in an OpenStack deployment, it is extremely important to have at least two physical Ethernet switches, two or more converged network adapters (CNAs) or NICs per host, and two or more NICs per storage controller. The network layout for the environment should be carefully planned, and detailed visual diagrams that display the connections for each port should be developed.

Note: The recommendations in this section apply generically to all hypervisors, OpenStack distributions, and NetApp storage systems. See the "Technical Reports and Documentation" section at the end of this document for other documents (for example, NetApp TR-4068: VMware vSphere 5 on NetApp Clustered Data ONTAP: Best Practices and NetApp TR-3848: Red Hat Enterprise Linux 6, KVM, and NetApp Storage: Best Practices Guide) that provide additional recommendations and best practices for specific deployment choices.

Security

The security of an OpenStack deployment is of paramount importance; therefore, security best practices should be followed for all nodes. Cloud administrators should refer to the hypervisor vendor's best practices to appropriately secure account settings, firewalls, and auxiliary services.

In addition to following the guidance in NetApp TR-3964: Clustered Data ONTAP Security Guidance, NetApp also strongly recommends using Secure Sockets Layer/Transport Layer Security (SSL/TLS) in an OpenStack deployment to provide protection for network traffic that might carry sensitive data such as user credentials or protected data.

Best Practice

NetApp strongly recommends that anyone deploying OpenStack thoroughly review the OpenStack Security Guide to understand and implement the community-authored security best practices.

10 Gigabit Ethernet (10GbE)

The NetApp FAS, E-Series, and flash array storage systems all support 10GbE. An advantage of 10GbE is its ability to reduce the number of network ports in the infrastructure, especially but not limited to blade servers. In addition, 10GbE can handle several VLANs simultaneously. It is a NetApp best practice to use 10GbE, especially for storage.

Segmentation of Management Traffic from Data Traffic

NetApp recommends separating storage network traffic from other networks. A separate network can be achieved by using separate switches or by creating a VLAN on shared switches. This network should not be routable to other networks. If switches are shared with storage and other traffic, it is imperative to confirm that the switches have adequate bandwidth to support the combined traffic load. Although the

storage VLAN should not be routable, other VLANs (such as those for management or VM traffic on the same switches) might be routable. VLANs allow multiple network functions to share a small number of high-speed network connections, such as 10GbE.

Routing and IP Storage Networks

Whenever possible, NetApp recommends configuring storage networks as single, isolated networks that are not routable. This method provides good performance and a layer of data security. Generally, this means that logical networks should be created on private address spaces as specified by IETF RFC 1918. NetApp recommends that you minimize the number of network hops between the storage subsystem and the host node.

Improving Network Performance and Resiliency with Link Aggregation

Link aggregation, standardized originally under IEEE 802.3AD but now as 802.1AX, refers to using multiple physical network connections to create one logical connection with combined throughput and improved redundancy. Several implementations of link aggregation are available, and they are known by different names: The VMware vSphere implementation is referred to as NIC teaming; Cisco trademarked the term EtherChannel; the NetApp implementation is called interface groups (ifgrps); before the release of Data ONTAP 8.0, the NetApp implementation was known as virtual interfaces. Other implementations might refer to bonding or trunking, although trunking has a different meaning in Cisco technology, referring to carrying multiple tagged VLANs on a link or channel between two switches. The term “load balancing” is also used in conjunction with link aggregation. In practice, the loads are usually not symmetrically balanced between links in a team, although multiple links can carry traffic.

Not all link aggregation implementations are alike or offer the same features. Some offer only failover from one link to another if one link fails. More complete solutions offer true aggregation, in which traffic can flow on two or more links at the same time. There is also the Link Aggregation Control Protocol (LACP), which allows devices to negotiate the configuration of ports into bundles. Failover-only configurations generally require no special capability or configuration of the switches involved. Aggregation configurations require compatible switches with the sets of ports configured for link aggregation.

NetApp interface groups are created on top of two or more ports. With interface groups, the logical interface (LIF) is associated with the interface group rather than with the underlying ports. Table 2 compares the three variations of interface groups.

Table 2) Data ONTAP interface group types.

Interface Group Type	Failover	Aggregation	Switch Configuration and Support Required?	Packet Distribution
Single mode	Yes	No	No	Single active link
Static multimode	Yes	Yes	Yes	IP, MAC, round robin, TCP/UDP port
Dynamic multimode (LACP)	Yes	Yes	Yes	IP, MAC, round robin, TCP/UDP port

Note: NetApp recommends using dynamic multimode if the switch supports LACP.

As specified in Table 2, single-mode interface groups send and receive traffic only on a single active link. Other links in the interface group remain unused until the first link fails. Failover groups offer an improved design for use with switch configurations that do not support link aggregation. Instead of configuring a single-mode interface group, the administrator can assign the ports to a failover group. LIFs that use these ports are set to use this failover group and have their failover policy set to `nextavail`, which causes them to usually prefer ports on the current node in the event of a port failure. Each LIF has a

different home port. Although traffic is not balanced, all links can carry traffic concurrently and can take over for each other in the event of a link failure.

Early switch implementations of link aggregation allowed ports of only a single switch to be combined into a team, even on many stackable switches. More recent switches offer technology that allows ports on two or more switches to become a single team. Switches are connected to each other either with interswitch links that might be 1GbE or 10GbE or through proprietary cables.

Whether a single switch or a pair of properly stacked switches is used, the configuration on the OpenStack and storage nodes is the same because the stacking technology makes the two switches look like one to the attached devices.

Best Practices

NetApp recommends the following best practices for link aggregation:

- Use switches that support link aggregation of ports on both switches.
- Enable LACP for switch ports connected to OpenStack nodes and Data ONTAP nodes.
- Use IP hashing on OpenStack nodes (use the `layer2+3` option for `xmit_hash_policy`).
- Use dynamic multimode (LACP) with IP hashing on Data ONTAP nodes.

For more information about link aggregation, refer to NetApp TR-3802: Ethernet Storage Best Practices.

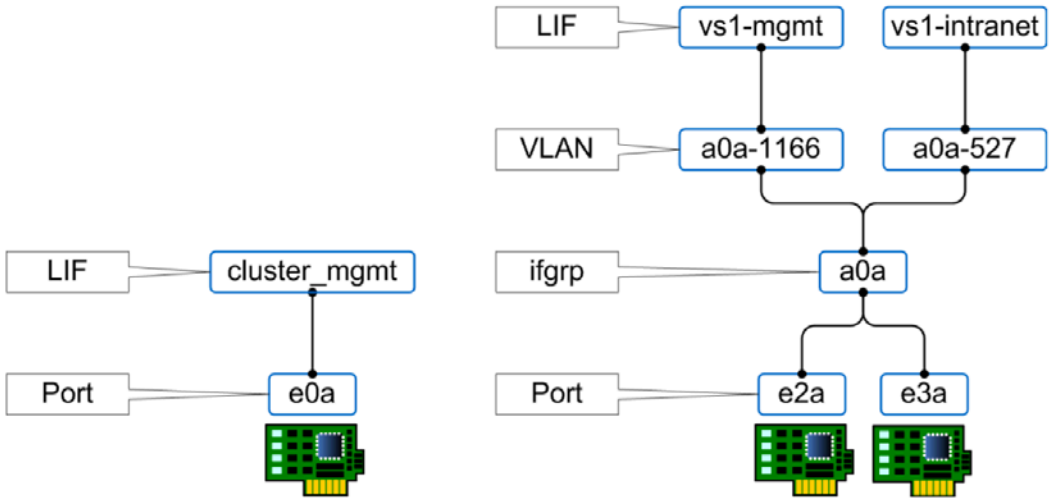
Network Time Protocol

All of the nodes in an OpenStack deployment must be configured to use NTP to avoid issues that arise from time skew. It is extremely important that the NTP service be highly available itself, although best practices for arranging that high availability are outside the scope of this document.

Data ONTAP Networking

The physical interfaces on a Data ONTAP node are referred to as ports. IP addresses are assigned to logical interfaces (LIFs). LIFs are logically connected to a port. As Figure 13 shows, physical ports can be grouped into interface groups (abbreviated as ifgrps). VLANs can be created on top of physical ports or interface groups. LIFs can be associated with ports, interface groups, or VLANs.

Figure 13) Simple and advanced examples of ports and LIFs.

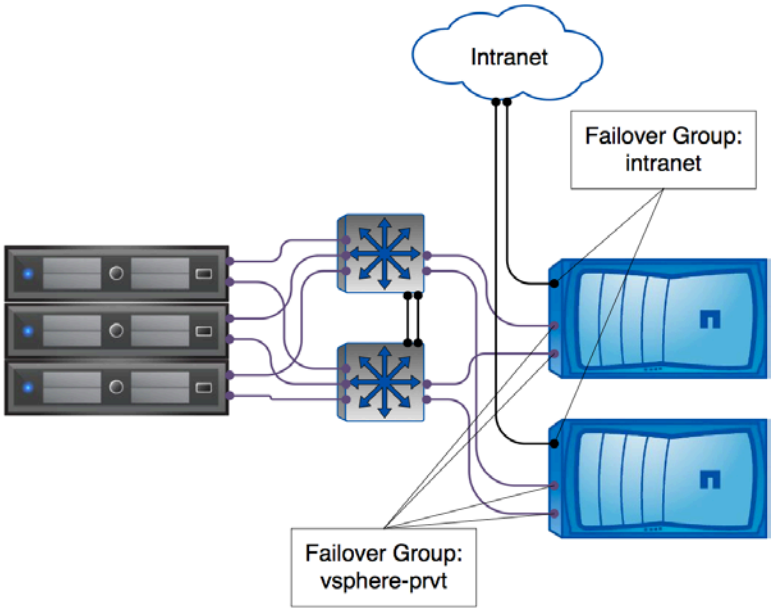


LIFs and ports have roles, which include cluster or node management, cluster (for traffic between nodes), intercluster (for NetApp SnapMirror® replication), and data.

From a solution perspective, data LIFs are further classified by how they are used by the servers and applications and by whether they are on private nonroutable networks, on corporate internal routable networks, or in a demilitarized zone. The NetApp cluster connects to these various networks through data ports; the data LIFs must use specific sets of ports on each node for traffic to be properly routed.

Some LIFs, such as the cluster management LIF and the data LIFs for NFS and CIFS, can fail over between ports in the same node or between nodes; if a cable is unplugged or a node fails, traffic continues to flow without interruption. Failover groups, such as those shown in Figure 14, are used to control the ports to which a LIF might fail over. If failover groups are not set up or are set up incorrectly, LIFs might fail over to a port on a wrong network, causing connectivity to be lost.

Figure 14) Failover groups.



Best Practices

- Make all Ethernet data ports, interface groups, and VLANs members of an appropriate failover group.
- Associate all NFS, CIFS, and management LIFs with the appropriate failover group.
- To keep network connectivity as simple as possible, use the same port on each node for the same purpose (assuming similar nodes with similar ports).

For information about using the NFS storage protocol with clustered Data ONTAP, refer to NetApp TR-4067: Clustered Data ONTAP NFS Best Practice and Implementation Guide.

iSCSI Networking Considerations

iSCSI is an Ethernet protocol that transports SCSI commands over an IP-based network. It functions similarly to FC in providing block storage to the initiator (or storage consumer) from the target (or storage provider); however, it uses Internet Protocol (IP) rather than FC as the transport. This leads to some differences; for example, FC has buffers that prevent frames from being dropped in the event of congestion. However, because iSCSI is IP based, it relies on SCSI to be tolerant of dropped Ethernet frames, and TCP retransmits when congestion or errors occur.

iSCSI uses standard network switches for transporting data, which makes the network a critical component of the storage architecture. If the network is not highly available and is not capable of providing enough throughput for VM storage activity, significant performance and availability issues can result. NetApp recommends connecting both the NetApp controllers in the cluster that provides iSCSI service and the OpenStack nodes that use iSCSI LUNs to more than one physical switch to provide redundancy in the event of a switch failure. The user should also work closely with the network administrator to make sure that the environment does not contain any severely oversubscribed uplinks that will be heavily used by iSCSI data connections. These bottlenecks can cause unpredictable performance issues and should be avoided if possible.

Production iSCSI traffic should always be connected through highly available network connections. HA can be provided through several methods, some of which also increase the available throughput to the iSCSI target.

Best Practice

NetApp recommends using dedicated iSCSI VLANs that are available to all participating hosts. This method isolates the unprotected storage traffic and should also provide a contiguous network subnet for connectivity.

iSCSI Networking Considerations for Clustered Data ONTAP

The Data ONTAP operating system can use multiple sessions across multiple networks. For the iSCSI storage network design, NetApp recommends using multiple physical network interfaces that are connected to multiple physical switches that use link aggregation to provide HA and increased throughput. For Data ONTAP controller connectivity, NetApp recommends configuring at minimum a single-mode interface group for failover with two or more links that are connected to two or more switches. Ideally, LACP or other link-aggregation technology (as previously described) should be used with multimode interface groups to provide HA and the benefits of link aggregation. The NetApp storage system is the iSCSI target, and many different initiators with many different IPs are connected to it, which significantly increases the effectiveness of link aggregation.

Clustered Data ONTAP SVMs can have many LIFs supporting multiple protocols. The NetApp best practice is to have an iSCSI LIF for the SVM on each physical node in the cluster. Clustered Data ONTAP and the storage client use asymmetric logical unit access (ALUA) to correctly choose the path that provides direct access to the LUNs. The client host discovers which paths are optimized by sending a status query to the iSCSI LUN host down each path. For the paths that lead to the clustered Data ONTAP node that directly owns the LUN, the path status is returned as `active/optimized`; for other paths, the status is returned as `active/nonoptimized`. The client prefers the optimized path whenever possible. Without ALUA, a path that traverses the cluster network can be selected as the primary path for data. This configuration still functions as expected (that is, all of the LIFs accept and process the LUN reads and writes); however, because it is not the optimal path, a small amount of additional latency is incurred from traversing the cluster network.

LIFs that are used for block protocols such as iSCSI cannot be migrated between nodes of the clustered Data ONTAP system. This is not the case with CIFS and NFS; with these protocols, the LIFs can be moved to any of the physical adapters that the SVM can access. In the event of failover, the takeover node becomes the new optimized path until giveback is performed. However, if the primary node is still active, but the LIF is inactive because of network failure or other conditions, the path priority remains the same. A volume move between nodes causes the node paths to be reevaluated and the client to select the new direct or optimized path as the primary path.

iSCSI Networking Considerations for E-Series

The NetApp E-Series storage controllers support HA through the use of ALUA and failover drivers that are configured on hosts. The primary failover driver used is the Device Mapper Multipath (DMMP) driver,

which is a generic framework for block devices provided by the Linux OS. The multipath function is provided by the combination of kernel modules and user space tools.

DMMP accomplishes the following tasks:

- Provides a single block device node for a multipathed logical unit
- Reroutes I/O to available paths during a path failure
- Revalidates failed paths as quickly as possible
- Configures the multipaths to maximize performance
- Reconfigures the multipaths automatically when events occur
- Provides DMMP features support to newly added logical units
- Provides device name persistence for DMMP devices under `/dev/mapper/`
- Configures multipaths automatically at an early stage of rebooting to permit the OS to install and reboot on a multipathed logical unit

NetApp E-Series storage controllers do not currently support link aggregation, so a highly available storage deployment depends on correctly configuring failover drivers to avoid multiple paths in the logical network topology between hosts and storage controllers. For information about how to configure the DMMP driver, refer to the NetApp E-Series Storage Systems Failover Drivers Guide.

Load Balancers

All of the REST APIs used by OpenStack services should be offered through a tier of redundant load balancers, configured to distribute incoming connections across all healthy API endpoints (as described in Table 1). Either software-based or hardware-based load balancers can be used with an OpenStack deployment.

The distribution algorithm configured on the load balancers should distribute the incoming workload efficiently across all valid service endpoints. NetApp recommends using a weighted-least-connections distribution algorithm, if present. Otherwise a standard round-robin distribution algorithm might be sufficient for use.

Best Practice

NetApp recommends configuring the health-checking mechanism of the deployed load balancers to perform service-level (HTTP) health checks rather than simply to verify that a TCP connection can be established. At the time of writing, OpenStack Object Storage (Swift) is the only service that exposes an explicit health-check endpoint (`/healthcheck`) that can be configured on a load balancer. All other services can be checked by issuing an HTTP `GET` request against the `/` URL.

Jumbo Frames

Jumbo frames are larger Ethernet packets that reduce the ratio of packet overhead to payload. The default Ethernet frame size or maximum transmission unit (MTU) is 1,500 bytes. With jumbo frames, MTU is typically set to 9,000 on end nodes, such as servers and storage, and to a larger value, such as 9,198 or 9,216, on the physical switches.

Jumbo frames must be enabled on all physical devices and logical entities from end to end to avoid truncation or fragmentation of packets with the maximum size. On physical switches, the MTU must be set to the maximum supported value, either as a global setting or policy option or on a port-by-port basis (including all ports used by OpenStack, the nodes of the Data ONTAP cluster, and/or the NetApp E-Series controllers), depending on the switch implementation. The MTU must also be set, and the same value must be used, on the hypervisor and on the physical ports or interface groups of each node. When problems occur, it is often because the hypervisor was not correctly configured for jumbo frames.

Best Practice

It is a NetApp best practice to use jumbo frames for Ethernet storage traffic, especially NFS. Standard frames require NFS datagrams to be broken up and reassembled at the destination. This causes a performance hit on both ends of the wire. In contrast, a jumbo frame allows NFS datagrams to be sent whole, removing the need to break them up and reassemble them.

Jumbo frames should be configured for storage networks and isolated, nonrouted VLANs that carry NFS, CIFS, and iSCSI data streams.

4 Database and Messaging Subsystems

This section describes the database and messaging subsystems and explains the in-memory key/value store.

4.1 Database Subsystem

OpenStack uses a SQL database to store the operational state for a cloud infrastructure; this state includes data representing the instance types that are available for use, instances in use, networks available, projects, and so forth. Each core OpenStack service stores its state and configuration information in the database.

MySQL and PostgreSQL are widely used for production OpenStack deployments. This section focuses on MySQL because most OpenStack deployments use it. It is the most tested for use with OpenStack and it is very well documented. NetApp recommends using the MySQL InnoDB storage engine of MySQL, which is a transaction-safe storage engine capable of performing rollback and crash recovery. MySQL meets the data security and multi-tenancy requirements of OpenStack cloud deployments.

The database leveraged by an OpenStack cloud can be deployed in different ways:

- By leveraging an existing MySQL infrastructure
- By installing and configuring it on the controller nodes
- By installing and configuring it on an isolated node or nodes

The MySQL infrastructure does not have to be placed on the OpenStack environment if a highly available instance of MySQL is available to reuse. You can make that decision based on your current environment or the clustering topology for other OpenStack services that you want to use.

This section is not meant to describe an all-inclusive, failure-proof HA environment, but rather to provide a conceptual architecture for the options that are available to protect the MySQL database. Although there are multiple methods for creating a highly available database service, this section focuses on a database replication and clustering architecture (active-passive) that protects the data and provides failover and a tightly coupled replication and clustering architecture (active-active) using Galera cluster with cluster control.

Best Practices

- Use the InnoDB storage engine of MySQL.
- Restrict database network traffic to the OpenStack management network.
- Configure the database to require SSL for database clients to protect data “on the wire.”

Storage Considerations

NetApp provides the ideal storage platform for MySQL enterprise databases that require HA and high performance. NetApp storage not only supports but also excels at protecting and serving data with all of the major storage protocols, including NFS and iSCSI. In support of enterprise data systems, NetApp

storage systems provide superior, cost-effective data protection, backup and recovery, availability, and administration through NetApp tools and features such as:

- Fast and efficient backup and recovery using NetApp Snapshot technology
- Quick cloning with NetApp FlexClone technology
- Data protection, scalability, and manageability with nondisruptive operations with NetApp clustered Data ONTAP

When deploying a MySQL database in an OpenStack deployment that uses NetApp storage, refer to NetApp TR-3657: Best Practices and Guidelines for MySQL, which provides recommendations and guidelines for NetApp storage and server settings for a MySQL environment.

Protocol Considerations

The protocol used (NFS, iSCSI, or FC) depends on other features desired for the HA configuration. The use of NFS provides the following benefits:

- Capability of resizing as required without downtime
- Easier manageability of FlexVol volumes and file systems
- Leveraging of advanced features of NetApp clustered Data ONTAP, such as single-file restores using NetApp SnapRestore® technology

When either NFS, iSCSI, or FC is used, no functionality of the NetApp features is lost.

Storage Design

The structure of the NetApp FlexVol volumes used to store the database is driven by backup, restore, and data protection requirements. The MySQL database should use dedicated FlexVol volumes for each of the database components. This design leverages the efficient backup and recovery capabilities of a MySQL database and provides the capability to use FlexClone and SnapMirror features of Data ONTAP.

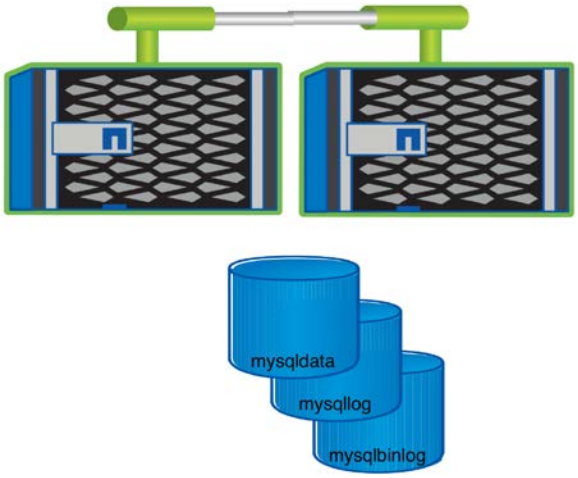
Table 3 lists the components of a recommended NetApp FlexVol volume storage design for specific MySQL database components.

Table 3) MySQL and NetApp logical design.

MySQL Database Component	FlexVol Volume Name	File System Path
Transaction log files	mysqllog	`/var/lib/mysqllog`
Binary log	mysqlbinlog	`/var/lib/mysqlbinlog`
Data directory	mysqldata	`/var/lib/mysqldata`

Figure 15 shows the layout of these components.

Figure 15) MySQL with NetApp FlexVol volume storage design.



Best Practice
NetApp recommends breaking out the MySQL database components into individual FlexVol volumes, allowing the appropriate MySQL binary log files to be applied for any necessary recovery after a restore of the data. For appropriate MySQL and NetApp settings, refer to NetApp TR-3657: Best Practices Guidelines for MySQL.

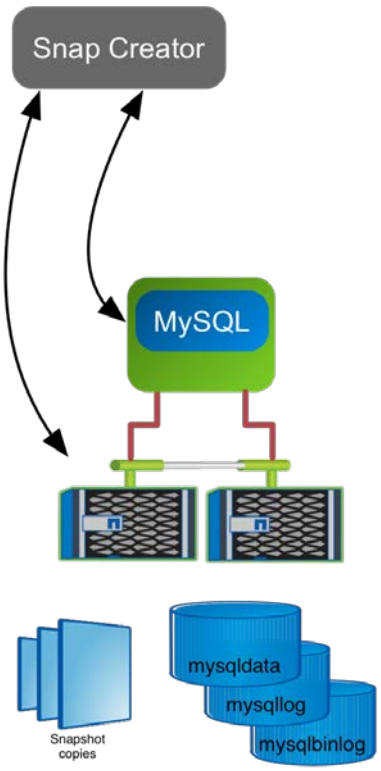
Backup and Recovery

The structure of the NetApp FlexVol volumes used to store the database provides the capability for efficient backup, recovery, and disaster recovery for the MySQL database. The storage design allows users to leverage the NetApp features provided by Data ONTAP more effectively.

NetApp Snapshot copies can offer efficient backups and restores for any database. The Snapshot copy creation is almost instantaneous, allowing customers to protect data with fast, space-efficient backups; however, the Snapshot copies should be coordinated with the MySQL database for an application-consistent backup. Because the NetApp Snapshot backup is created instantaneously, there is no interruption of service for an online MySQL backup that is made using NetApp Snapshot copy technology.

Of the different ways that NetApp offers to create a backup or restore the MySQL database, the most efficient method is to use the NetApp Snap Creator™ Framework tool with the MySQL plug-in. This tool allows you to back up, restore, and mirror the database, as shown in Figure 16.

Figure 16) MySQL with NetApp FlexVol Snapshot copies.



Best Practices
• Create application-consistent backups with NetApp Snapshot copies by flushing the MySQL cache to disk. • Use NetApp tools, such as Snap Creator, as a mechanism to execute efficient backups and restores of the MySQL database. • Manage the clustering and replication carefully when executing a restore. The storage-based NetApp Snapshot copy is not application aware. For more information about this configuration, refer to NetApp TR-3601: Online MySQL Backup Using NetApp Snapshot Technology.

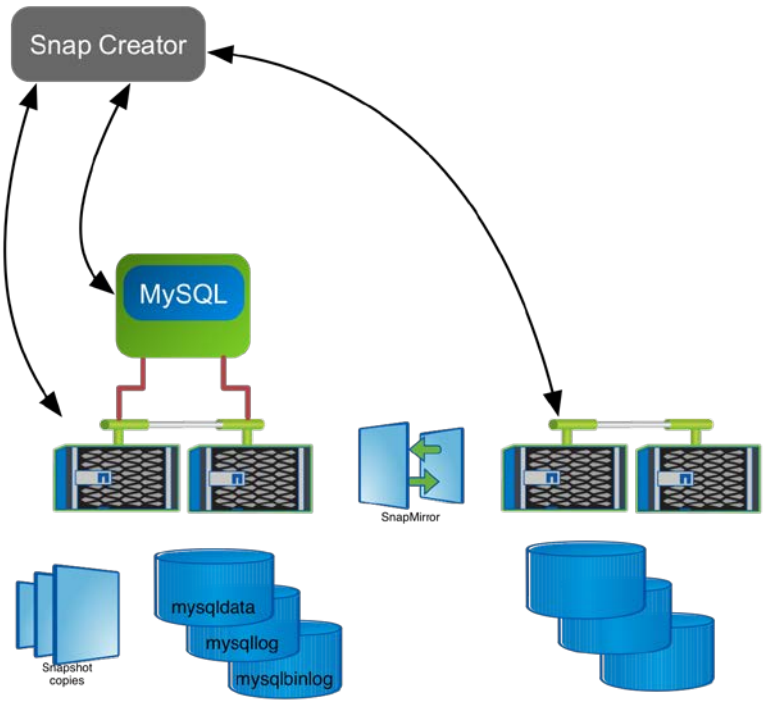
Disaster Recovery

Although application methods exist for protecting the MySQL database for HA purposes, NetApp SnapMirror technology can also be used as part of a disaster recovery methodology. SnapMirror has been used in other use cases to populate a replica of the database or create a development test environment based on a production environment.

SnapMirror provides the capability to maintain a remote data center in an off-site location where data can be stored to keep it available even during a failover triggered by a catastrophic event instead of by a hardware failure.

The Snap Creator product can also be used to manage the updating of a SnapMirror relationship, as shown in Figure 17.

Figure 17) MySQL with SnapMirror.



Active-Passive Configuration

This section describes the options of basing active-passive configurations on Pacemaker and Corosync clustering with Percona Replication Manager (PRM) or on Distributed Replication Block Device (DRBD) with Pacemaker and Corosync.

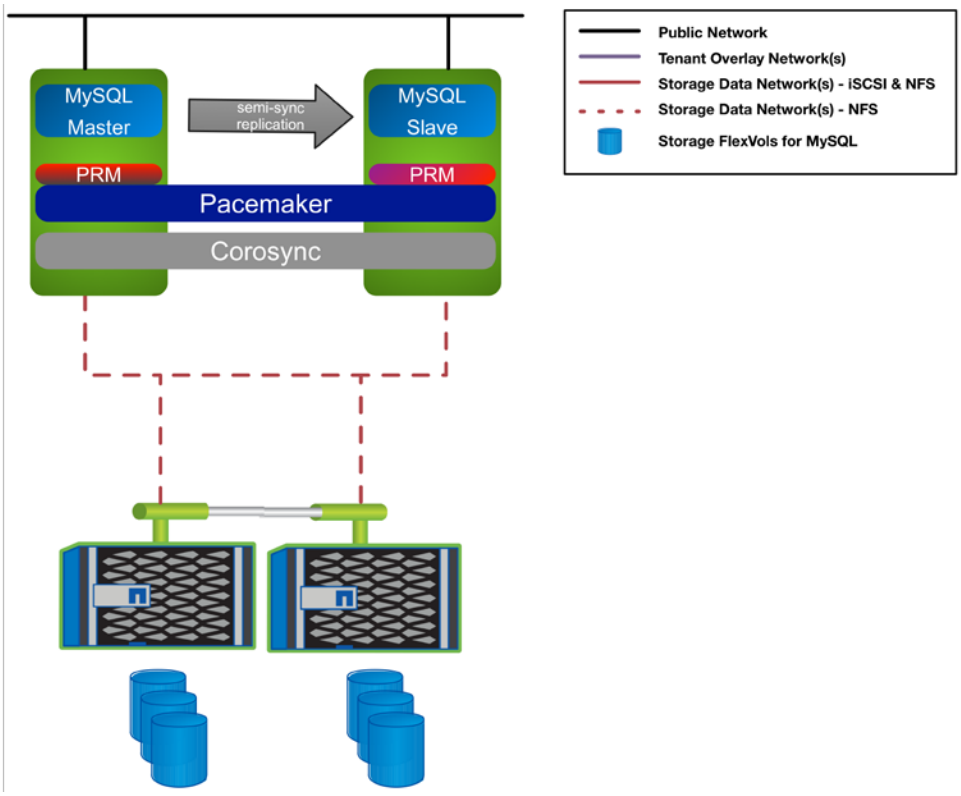
Pacemaker and Corosync Clustering with PRM

Corosync with Pacemaker is a clustering technology that can manage a MySQL environment. As Figure 18 shows, Pacemaker and Corosync combine to provide the clustering layer that sits between the services and the underlying hosts and OSs. Pacemaker is responsible for starting and stopping services, confirming that they are running on one host, delivering HA, and avoiding data corruption. Corosync provides the underlying messaging infrastructure between the nodes that enables Pacemaker to do its job.

Another component for this configuration is the Percona Replication Manager (PRM). PRM is a framework using the Linux HA resource agent called Pacemaker that manages replication and provides automatic failover. PRM is a specific MySQL resource agent and configuration convention used to manage the master-slave replication. PRM can assign the slave replication partner the master status and fail over to that database.

Best Practices
• Use the NFS protocol for the backend storage of the MySQL database. • Use semisynchronous or asynchronous replication, depending on your SLA. • Follow guidelines provided for your MySQL storage design to leverage NetApp Snapshot technology to execute efficient backup-and-restore functionality.

Figure 18) MySQL HA with clustering and PRM.



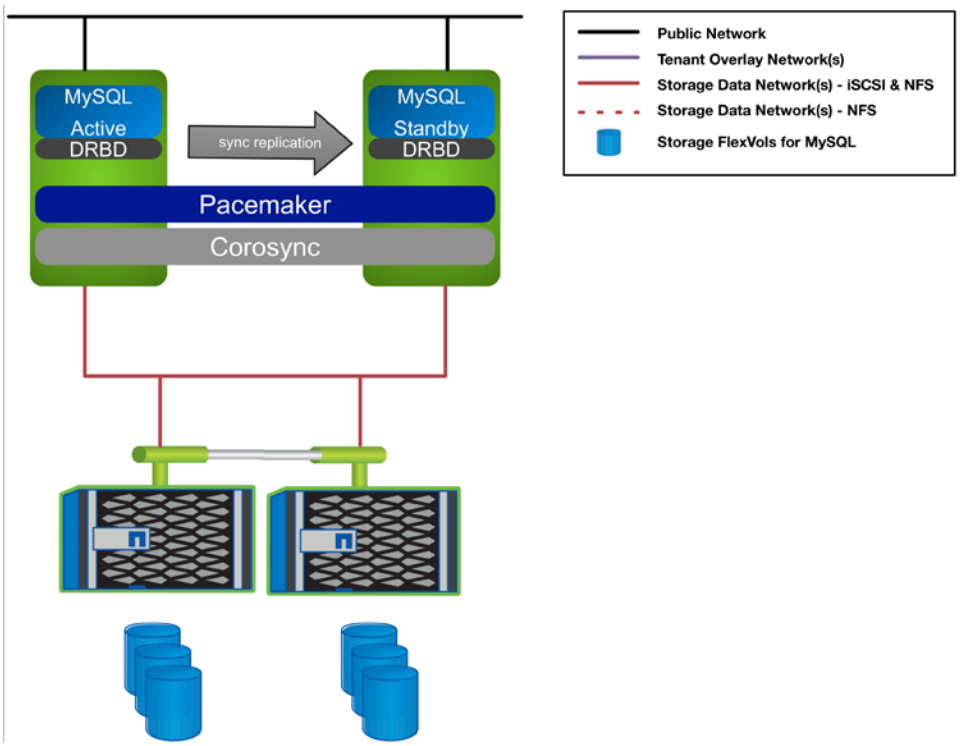
Distributed Replication Block Device

A method that uses another form of replication for an active-passive HA infrastructure is to combine Distributed Replication Block Device (DRBD) with Pacemaker and Corosync, as shown in Figure 19. This method provides an end-to-end stack of mature open-source technology for protecting the MySQL database of the OpenStack ecosystem. It provides an automatic failover capability (host clustering) and a synchronous replication model between DRBD nodes without risk of losing data committed to the database.

The combination of DRDB, Pacemaker, and Corosync provides a protected database with a clustering layer that sits between the services and the underlying OSs to provide a typical host clustering mechanism. This type of architecture provides both data protection through synchronous replication and the ability to provide automatic failover for service continuity.

Best Practice
When using DRBD, use the iSCSI protocol with LUNs for the backend storage of the MySQL database.

Figure 19) MySQL HA with DRBD.



Active-Active Configuration

This section describes the method of creating an active-active configuration through synchronous replication with Galera clustering.

Synchronous Replication with Galera Clustering

One method of creating an active-active configuration for the MySQL component of an OpenStack ecosystem is to combine MySQL with Galera. This deployment pattern is widely used in the OpenStack community.

Galera replication is a synchronous multimaster replication plug-in for InnoDB. It differs from the regular MySQL replication in addressing a number of issues, including conflict resolution when writing on multiple masters, replication lag, and having slaves that are out of sync with the master. Users do not have to know to which server they can write (the master) or from which servers they can read (the slaves). An application can write to any node in a Galera replication cluster, and transaction commits (RBR events) are then applied on all servers through a certification-based replication.

Galera replication is a plug-in to the MySQL deployment, and it enables the true master-master setup for the InnoDB. With Galera, all nodes are considered the master node, without the concept of a slave node. This can provide data consistency across all nodes because the replication has parallel applier threads. With this type of architecture, applications can read and/or write from any node of the cluster. Transactions are synchronously committed on all nodes. If a node fails, the other nodes continue to operate, and the data is kept in sync. At a minimum, three nodes are required for the MySQL Galera cluster because Galera is quorum based, which requires at least three nodes in order to have a majority vote.

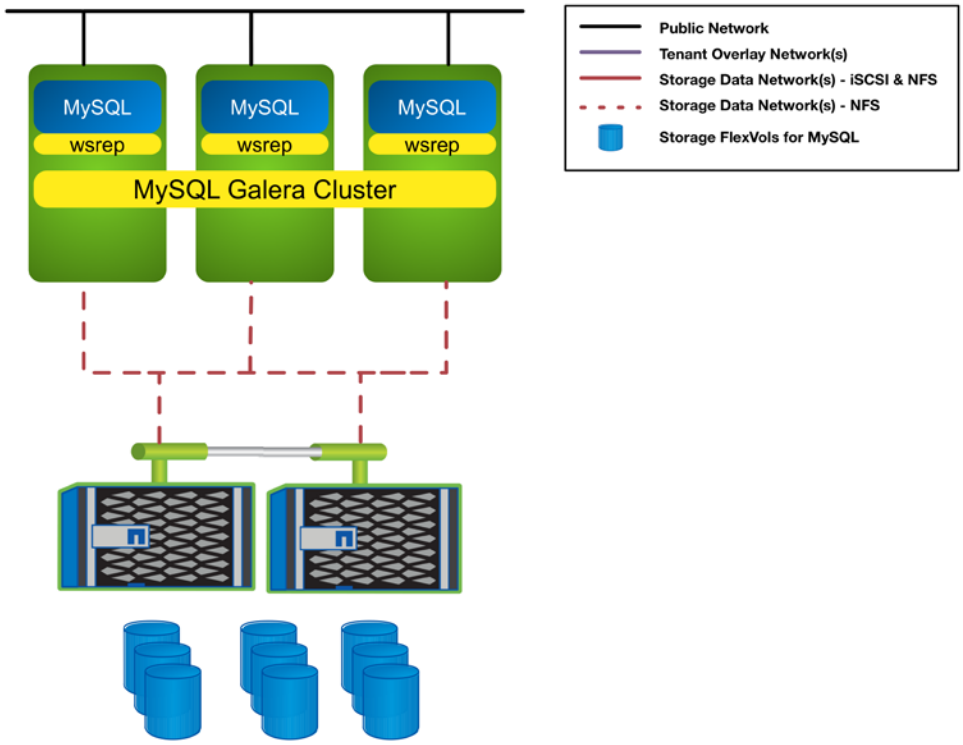
Important

Several uses of SQL statements leverage the `SELECT … FOR UPDATE` paradigm. Galera does not support `SELECT … FOR UPDATE` statements because it has no concept of cross-node locking of records and results are nondeterministic.

A solution to address this issue is to direct writes to a single cluster member and distribute reads across all cluster nodes. Splitting the traffic can be achieved through the use of the MySQL Proxy solution.

Figure 20 shows the logical architecture for the use of a Galera cluster with Data ONTAP in an OpenStack deployment.

Figure 20) MySQL HA with Galera clustering and replication.



Best Practices

- Use the NFS protocol for the backend storage of the MySQL database to take advantage of the flexibility of NetApp's storage technology.
- Follow the Galera configuration recommendations defined in the MySQL Galera cluster documentation.
- Follow guidelines provided for your MySQL storage design to leverage NetApp Snapshot technology copies to execute efficient backup-and-restore functionality.
- Deploy a minimum of three nodes for the MySQL Galera cluster because Galera is a quorum-based system that requires at least three nodes to achieve quorum.
- NetApp highly recommends that database clients access MySQL Galera instances through a load balancer.

MongoDB for Ceilometer

Ceilometer collects metering data (for example, CPU utilization and network bandwidth data) that can be used for billing purposes with OpenStack. The database most widely deployed with Ceilometer is MongoDB. Although there are drivers for other database backends, the storage driver for MongoDB is considered mature; therefore, NetApp recommends it for production use.

MongoDB Deployment Architecture

NetApp recommends deploying a minimal MongoDB replica set (two Mongo DB instances) and using automatic failover. The MongoDB can be configured to have multiple replica sets, as shown in Figure 21. For example, one can be local to the same availability zone as the primary MongoDB, with another replica in a different availability zone. The MongoDB replica sets are similar to those of a traditional master-slave replication configuration. In this configuration, a master node accepts the write(s), and one or more slave nodes replicate the write operations. With the use of MongoDB shards, each shard contains a replica of the primary data. Shards are not accessed directly through a client; clients access the MongoDB instances that act as a router for data access requests. If the infrastructure is large, MongoDB sharding helps maintain performance and scale-out for larger deployments.

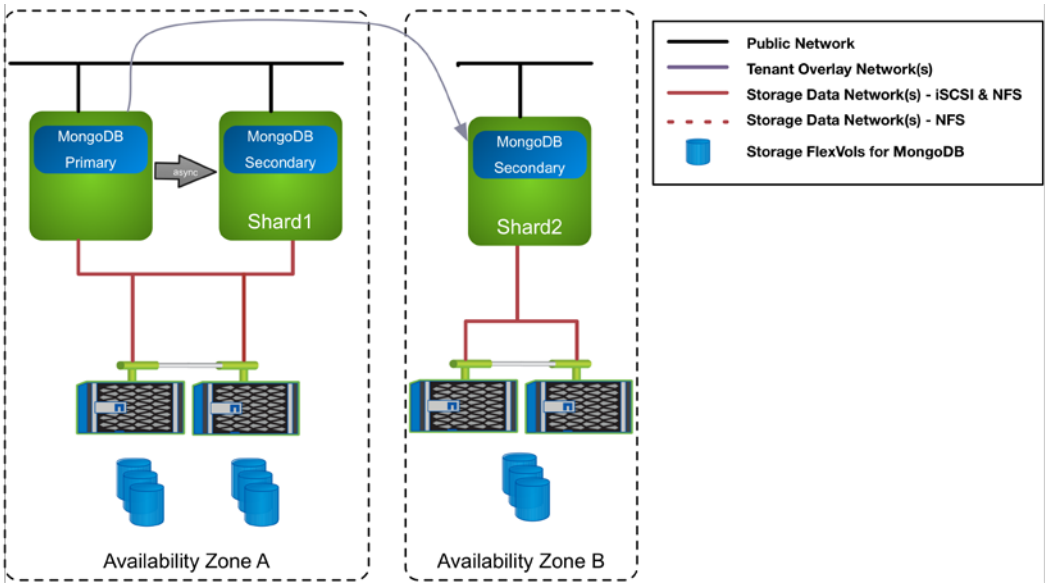
The scalability of NetApp clustered Data ONTAP is a complement to a MongoDB sharded cluster. Cloning can be used to create the new replica sets in the same Data ONTAP cluster. NetApp also recommends using SnapMirror when a replica set is required in a different data center. MongoDB can reside on separate nodes or can coreside with other elements of the OpenStack architecture.

Best Practices

- Always use replica sets.
- Run MongoDB on 64-bit systems to take advantage of memory addressability.
- Identify a minimum of three configuration servers to be deployed when sharding is used.
- Maintain multiple replica sets, which will be a complete copy within each shard. NetApp recommends keeping a replica set in a secondary data center for HA.
- Use different FlexVol volumes for the journal and data volumes of MongoDB for performance, ease of backup, and the ability to quickly create replica sets with NetApp Snapshot technology.
- Use the iSCSI protocol for backend storage of the MongoDB when journaling is in use. Journaling is the recommended best practice with MongoDB, and NFS is not supported with the write-ahead journaling feature that supports crash recovery.
- Follow guidelines provided for your MySQL storage design to leverage NetApp Snapshot technology to execute efficient backup-and-restore functionality.

For more information about NetApp and MongoDB, refer to NetApp WP-7188: NoSQL Technologies and NetApp.

Figure 21) MongoDB with replica sets.



4.2 Messaging Subsystem

Most OpenStack services are composed of individual processes that fulfill different subsets of capability that are combined to deliver a higher-level service. These processes require a mechanism to communicate with the other processes running in the context of a service. Given the modular, distributed, scale-out design goal of OpenStack, a message-oriented middleware system based on the Advanced Message Queuing Protocol (AMQP) is leveraged to provide the interprocess communication medium.

RabbitMQ

Although support exists in OpenStack for several AMQP brokers, RabbitMQ is the most widely deployed middleware platform in OpenStack environments. This section covers the key items to consider when deploying RabbitMQ in a highly available topology.

A RabbitMQ broker is a logical grouping of one or more nodes, each running the RabbitMQ application and sharing users, virtual hosts, queues, exchanges, and so forth. The collection of nodes is referred to as a cluster. The composition of a cluster can be altered dynamically.

As with the database subsystem, the RabbitMQ cluster leveraged by an OpenStack deployment can be deployed in different ways:

- Leverage an existing RabbitMQ cluster
- Install and configure it on the controller nodes
- Install and configure it on an isolated set of nodes

Best Practice
Although RabbitMQ clusters can tolerate the failure of individual nodes, RabbitMQ clustering does not tolerate network partitions well. Be sure to leverage a redundant management network as described in section 3, “Networking Best Practices.”

By default, RabbitMQ message queues reside only on the node that created them, although they are visible and reachable from all nodes. Although RabbitMQ queues can be configured to be durable (that is, persisted to disk), it is generally considered preferable to persist queue contents only to local memory and to rely on the queue-mirroring function of RabbitMQ instead. By replicating the contents of message

queues across all nodes in the RabbitMQ cluster, clients obtain resiliency to individual node failure in the cluster, without the performance penalty of persisting each message to disk.

Best Practice
NetApp recommends deploying RabbitMQ in a clustered topology with mirrored queues to provide a highly available messaging subsystem for OpenStack deployments.

4.3 In-Memory Key/Value Store

At several places in OpenStack, an in-memory key/value store can be leveraged to improve performance by caching data in a low-latency cache and reducing the number of queries that must be performed against a backend data source. Memcached is an open-source memory caching system that is widely used as part of OpenStack deployments.

High Availability of Memcached

All OpenStack services that leverage memcached support the declaring of multiple memcached endpoints. It is important to note that the data in a single memcached instance is not replicated to other instances. In the case of a memcached instance failure, the client simply fails over and leverages a secondary instance. Although the secondary instance might not have the data that is being requested by the client, at worst a full query to the datastore of record occurs and places the current record into the secondary instance.

Best Practice
NetApp recommends deploying memcached in OpenStack installations to optimize the performance characteristics of various services.

5 OpenStack Compute Storage Service (Nova)

The OpenStack Compute Service (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. Nova provides management services for VM instances across a fleet of potentially disparate hypervisors or other virtualization mechanisms such as containers. Nova depends heavily on the `libvirt` virtualization API to offer its management services.

Nova is typically deployed in conjunction with other OpenStack services (for example, block storage, object storage, and image) as part of a larger, more comprehensive cloud infrastructure.

Table 4 describes the processes that make up the Nova service.

Table 4) Overview of Nova processes.

Process Name	HA Strategy	Location	Description
`nova-api`	Active-active	Controller node	A Web Server Gateway Interface (WSGI) application that accepts and validates REST (JSON or XML) requests from clients regarding instances
`nova-compute`	None: • KVM • Hyper-V • LXC	Compute node: • KVM • Hyper-V • LXC	Manages the communication between Nova and the hypervisors and guest VMs

Process Name	HA Strategy	Location	Description
	• Xen • VMware ESXi	Controller node: • VMware vCenter	
	Active-passive: • VMware vCenter	Guest VM on host: • Xen • VMware ESXi	
`nova-conductor`	Active-active	Controller node	Responsible for interacting with SQL database to manage instance metadata
`nova-cert`	Active-active	Controller node	Manages x509 certificates that are required for certain EC2-compatibility use cases
`nova-scheduler`	Active-active	Controller node	Determines on which compute node a VM should launch based on the request criteria and available resources
`nova-novnc`	Active-active	Controller node	Hosts a browser-based virtual network computing (VNC) client for end users to access the console interface for their instances
`nova-xvpvncproxy`	Active-active	Controller node	Hosts a traditional VNC endpoint for end users to access the console interface for their interfaces
`nova-consoleauth`	Active-passive	Controller node	Leveraged by both the `nova-novnc` and `nova-xvpvncproxy` processes to manage token authentication for console access

5.1 High Availability of Nova Processes

All Nova processes communicate with one another over the messaging subsystem through the AMQP protocol. All persistent state (that is, instance metadata) managed by the Nova service is stored in its backing SQL database, so most of the Nova processes are essentially stateless and can be scaled out across multiple controller nodes to provide a highly available Nova service; the two exceptions are `nova-compute` and `nova-consoleauth`.

`nova-compute` manages communication between the hypervisor/virtualization system and the rest of the Nova components. There should be one `nova-compute` process for each hypervisor host deployed in an OpenStack instance. The placement of the `nova-compute` process depends on the type of hypervisor:

- **KVM, Hyper-V, and LXC.** Each instance of `nova-compute` runs coresident with the hypervisor host. The availability of the `nova-compute` process is tied directly to the availability of the host for which it manages.
- **VMware ESXi, Xen.** Each instance of `nova-compute` runs in a guest running on the hypervisor host. The availability of the `nova-compute` process is tied directly to the availability of the guest VM and host for which it manages.
- **VMware vCenter.** Each instance of `nova-compute` runs on a host external to the vCenter or ESX deployment. NetApp recommends running `nova-compute` on the controller node in an active-

passive deployment style and managing the availability of the `nova-compute` process by a resource manager so that at least one instance is operational at all times.

The `nova-consoleauth` process manages the token-based authentication for access to the console of VM instances in a Nova deployment.

Best Practice
The current implementation of `nova-consoleauth` does not make its state available for other instances of `nova-consoleauth` to access to create a highly available service; therefore, `nova-consoleauth` must be deployed in an active-passive style.

5.2 High Availability of Virtual Machine Instances

The availability of VM instances running in a Nova deployment is inherently tied to the availability of the underlying hypervisor on which the VM runs. If an outage can be planned, Nova provides sufficient capabilities (dependent on the hypervisor deployed) to migrate a VM instance from one host to another without disruption to the guest. In the event of an unplanned outage, Nova has the ability to evacuate the state of all VMs that were running on a host to other hosts in the deployment. User data in ephemeral disks cannot be recovered unless the disks are stored on shared storage, as described in Section 5.3.

5.3 Storage Considerations

An important implementation choice when deploying Nova is deciding where instances are stored on the hypervisor's disk. Although this might normally point to locally attached storage, that arrangement would prohibit the ability to support live migration of instances between compute nodes. By specifying a directory that is a mounted NFS export from a NetApp FlexVol volume, it is possible to support live migration of instances because their root disks are on shared storage that can be accessed from multiple hypervisor nodes concurrently.

Best Practice
If live migration of VMs is a functional requirement of an OpenStack deployment, NetApp strongly recommends using an NFS export from a FlexVol volume as the storage for ephemeral disks for instances. Deduplication of this FlexVol volume can also provide storage efficiency benefits.

6 OpenStack Block Storage Service (Cinder)

The OpenStack Block Storage service manages persistent block storage resources. In addition to acting as auxiliary persistent storage, a Glance image can be copied into a Cinder volume for Nova to use as a bootable, persistent root volume for an instance. The Block Storage service was originally a Nova component called `nova-volume`, but it emerged as an official, independent project in the Folsom release. Cinder is conceptually similar in function to the well-known Amazon Elastic Block Storage offering.

Cinder is typically deployed in conjunction with other OpenStack services (for example, Compute, Object Storage, and Image) as part of a larger, more comprehensive cloud infrastructure. This is not an explicit requirement because Cinder has been successfully deployed as a standalone solution for block storage provisioning and lifecycle management.

Figure 22 shows the logical architecture of Cinder used in conjunction with Nova. As a management service, Cinder controls the provisioning and lifecycle management of block storage volumes. It does not reside in the I/O (data) path between the hypervisor and the storage controller.

Figure 22) Logical architecture of Cinder and Nova.

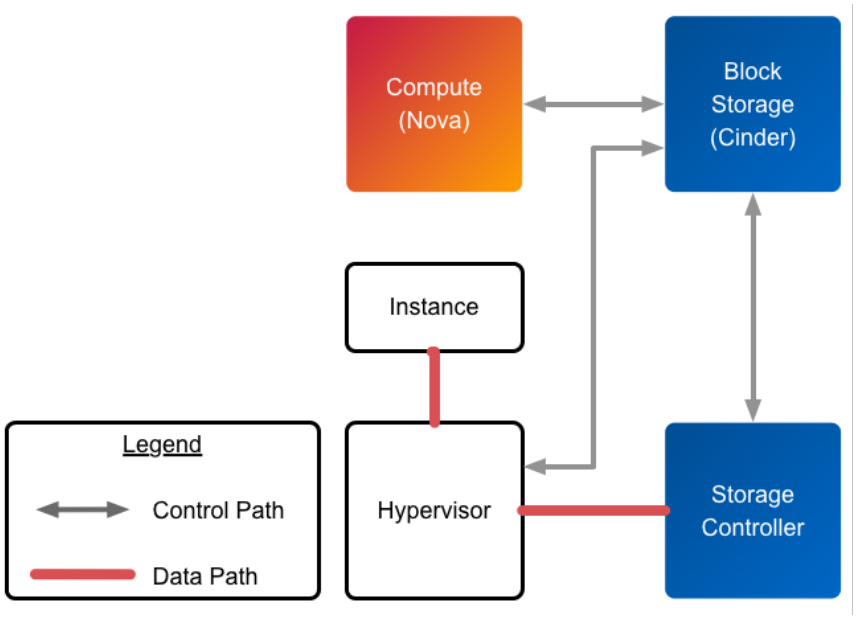


Table 5 describes the four processes that make up the Cinder service.

Table 5) Overview of Cinder processes.

Process Name	HA Strategy	Location	Description
`cinder-api`	Active-active	Controller node	A WSGI application that accepts and validates REST (JSON or XML) requests from clients regarding block storage volumes
`cinder-backup`	Active-active	Controller node	Handles the interaction with a backup target (for example, OpenStack Object Storage Service [Swift] or NFS) when a client requests a volume backup to be created or restored
`cinder-scheduler`	Active-active	Controller node	Determines which backend should serve as the destination for a volume creation or movement request **Note:** It maintains nonpersistent state for backends (for example, available capacity, capabilities, and supported extra specs) that can be leveraged when making placement decisions. The algorithm used by the scheduler can be changed through Cinder configuration.
`cinder-volume`	Active-active	Controller node	Accepts requests from other Cinder processes and serves as the operational container for Cinder drivers **Note:** This process is multithreaded and typically has one thread of execution per Cinder backend, as defined in the Cinder configuration file.

6.1 High Availability of Cinder Processes

As with the other OpenStack services, all of the Cinder processes communicate with each other over the messaging subsystem described in Section 4.2 and persist their state in a SQL database, as described in Section 4.1. Therefore, the Cinder processes are essentially stateless and can be scaled out across multiple controller nodes to provide a highly available block storage service.

6.2 Cinder Backup Service

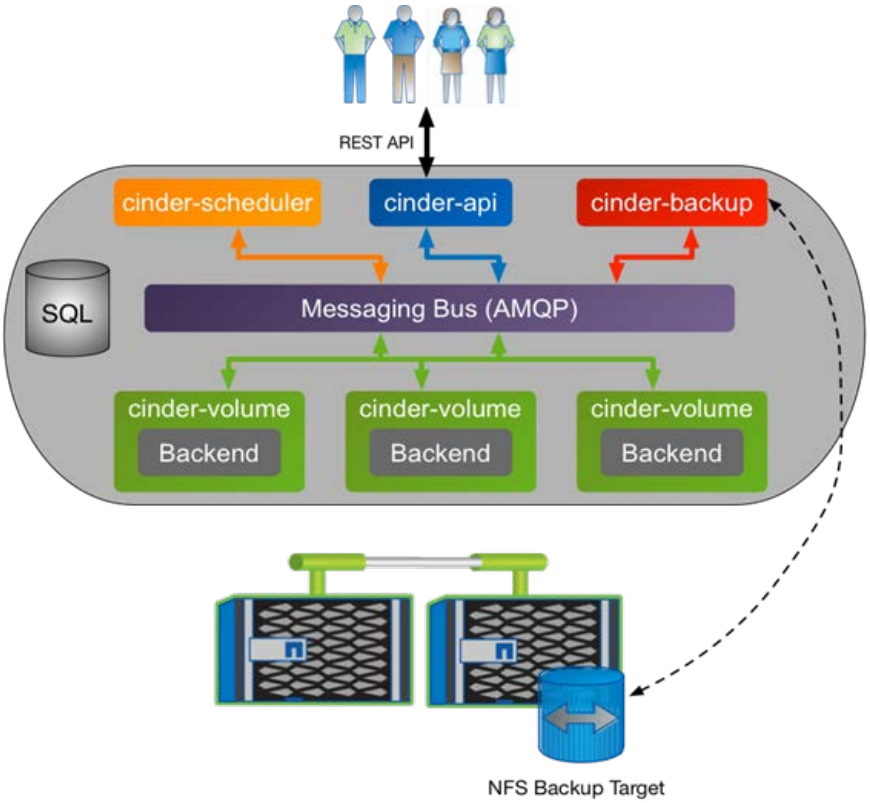
The OpenStack Block Storage service provides an interface to create a backup of Cinder volumes. Volume backups are full copies of persistent Cinder volumes in a backup repository. The Cinder Backup service can also restore from the backup repository to the original Cinder volume that was created or to a new Cinder volume. Cinder backups can be created, restored, deleted, and listed.

Before the Kilo release, the only supported backup stores were instances of OpenStack Object Storage Service (Swift), Ceph, or integration with Tivoli. By default, Swift is used for the backup repository. In the Kilo release, NetApp contributed a new driver for using an NFS export as a backup repository.

The NFS backup repository can be a NetApp FlexVol volume exported from a clustered Data ONTAP or Data ONTAP 7-Mode storage system to the controller nodes providing the Cinder Block Storage service. By using the NFS backup driver, you can also use NetApp SteelStore backup appliances as a repository.

When using a NetApp FlexVol volume as a Cinder backup repository, you can use Data ONTAP technology outside of the Cinder Backup service. Features such as SnapMirror, Snapshot copies, or FlexClone can be used to facilitate additional functionality and solutions with the Cinder Backup service. Figure 23 shows the Cinder Backup architecture.

Figure 23) Logical architecture of Cinder Backup service.



6.3 Concept Map for Clustered Data ONTAP and Cinder

The information in this section maps concepts between clustered Data ONTAP and Cinder. The two offerings share much of the same terminology.

Cinder Backends and Storage Virtual Machines (SVMs)

SVMs contain data volumes and one or more LIFs through which they serve data to clients. SVMs can contain either one or more of the data volumes known as FlexVol volumes or a single data volume that is based on an Infinite Volume. A single Infinite Volume can scale to greater than 20PB of raw capacity.

SVMs securely isolate the shared virtualized data storage and network, and each SVM appears to clients as a single dedicated SVM. Each SVM has a separate administrator authentication domain and can be managed independently by its SVM administrator. In a cluster, SVMs facilitate data access. A cluster must have at least one SVM to serve data. SVMs use the storage and network resources of the cluster. However, the volumes and LIFs are exclusive to the SVM. Multiple SVMs can coexist in a single cluster without being bound to any node in a cluster. However, they are bound to the physical cluster on which they exist.

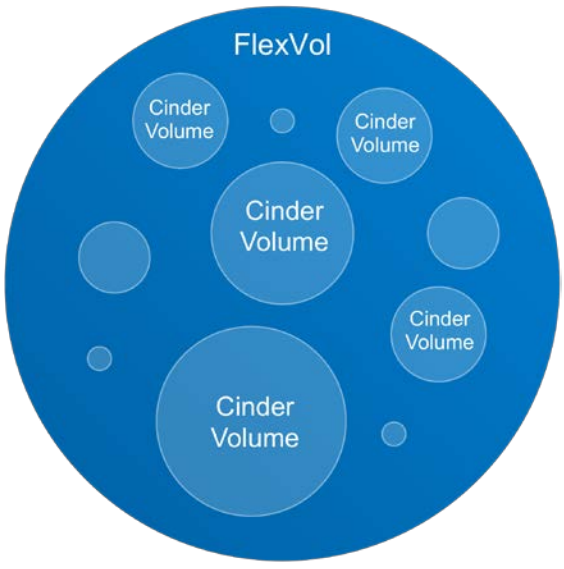
Best Practice

When Cinder is deployed with clustered Data ONTAP, NetApp recommends that each Cinder backend refer to a single SVM in a cluster. Although the driver can operate without the explicit declaration of a mapping between a backend and an SVM, a variety of advanced functionality (for example, volume type extra specs) will be disabled.

Cinder Volumes and FlexVol Volumes

Data ONTAP FlexVol volumes (commonly referred to as volumes) and OpenStack Block Storage volumes (commonly referred to as Cinder volumes) are not semantically analogous. A FlexVol volume is a container of logical data elements (for example, files, NetApp Snapshot copies, clones, LUNs, and so forth) that is abstracted from physical elements (for example, individual disks and RAID groups). A Cinder volume is a block device. Most commonly, these block devices are made available to OpenStack Compute instances. NetApp's various driver options for deployment of Data ONTAP as a provider of Cinder storage place Cinder volumes, Cinder snapshots, and clones in FlexVol volumes. Figure 24 shows that a FlexVol volume is a container of multiple Cinder volumes.

Figure 24) Cinder volumes versus FlexVol volumes.



Important
The FlexVol volume is an overarching container for one or more Cinder volumes.

A Cinder volume has a different representation in Data ONTAP when stored in a FlexVol volume, depending on the storage protocol that is used with Cinder:

- **iSCSI.** When the iSCSI storage protocol is used, a Cinder volume is stored as an iSCSI LUN.
- **NFS.** When the NFS storage protocol is used, a Cinder volume is a file on an NFS export.

Cinder Snapshots and NetApp Snapshot Copies

A NetApp Snapshot copy is a point-in-time file system image. Low-overhead NetApp Snapshot copies are made possible by the unique features of the NetApp WAFL® (Write Anywhere File Layout) storage virtualization technology that is part of Data ONTAP. The high performance of the NetApp Snapshot copy makes it highly scalable. A NetApp Snapshot copy takes only a few seconds to create, typically less than one second regardless of the size of the volume or the level of activity on the NetApp storage system. After a Snapshot copy is created, changes to data objects are reflected in updates to the current version of the objects, as if NetApp Snapshot copies did not exist. Meanwhile, the NetApp Snapshot version of the data remains completely stable. A NetApp Snapshot copy incurs no performance overhead; users can comfortably store up to 255 NetApp Snapshot copies per FlexVol volume, all of which are accessible as read-only and online versions of the data.

NetApp Snapshot copies are made at the FlexVol level, so they cannot be directly leveraged in an OpenStack context. That is because when a user of Cinder requests a snapshot, it is taken of a particular Cinder volume (not of the containing FlexVol volume). Because a Cinder volume is represented as either a file on NFS, an iSCSI LUN, or an FC LUN, Cinder snapshots are created through the use of NetApp FlexClone technology. By leveraging FlexClone technology to provide Cinder snapshots, it is possible to create thousands of Cinder snapshots for a single Cinder volume.

FlexClone files, or FlexClone LUNs and their parent files, or LUNs that are present in the FlexClone volume continue to share blocks in the same way that they do in the parent FlexVol volume. In fact, all FlexClone entities and their parents share the same underlying physical data blocks, minimizing physical disk space usage.

Best Practice
When Cinder is deployed with Data ONTAP, Cinder snapshots are created by leveraging the FlexClone feature of Data ONTAP. Therefore, a license option for FlexClone must be enabled.

6.4 Storage Considerations for Clustered Data ONTAP

This section explains storage considerations related to clustered Data ONTAP, including the storage protocol, the storage service catalog, and NetApp OnCommand® Workflow Automation (WFA).

Storage Protocol

When clustered Data ONTAP is deployed with Cinder, NFS, iSCSI, and FC are all valid choices for the storage protocol. They all support the same Cinder features, and they support Cinder snapshots and cloning to similar degrees, as well as other storage efficiency, data protection, and HA features.

iSCSI

The following guidelines apply for iSCSI:

- The current maximum number of iSCSI LUNs per NetApp cluster is either 8,192 or 49,152, depending on the FAS model number. (For detailed information about a particular model, refer to the NetApp

Hardware Universe.) Cinder can be configured to operate with multiple NetApp clusters through multi-backend support to increase this number for an OpenStack deployment.

- LUNs consume more management resources, and some management tools also have limitations on the number of LUNs.
- When Cinder is used independently of OpenStack Compute, the use of iSCSI is essential to provide direct access to block devices. The Cinder driver used in conjunction with NFS relies on libvirt and the hypervisor to represent files on NFS as virtual block devices. When Cinder is used in bare-metal or nonvirtualized environments, the NFS storage protocol is not an option.

NFS

The following guidelines apply for NFS:

- The maximum number of files in a single FlexVol volume exported through NFS depends on the size of the FlexVol volume; a 1TB FlexVol volume can have 33,554,432 files (assuming 32,000 inodes). The theoretical maximum of files is roughly 2 billion.
- NFS drivers require support from the hypervisor to virtualize files and present them as block devices to an instance.
- As of the Icehouse release, the use of parallel NFS (pNFS) is supported with the NetApp unified driver, providing enhanced performance and scalability characteristics.
- There is no difference in the maximum size of a Cinder volume, regardless of the storage protocol chosen (either a file on NFS or an iSCSI LUN is 16TB).
- When a VMware hypervisor environment is used, Cinder volumes can be attached to VMware guest VMs only over iSCSI because there is no support for mounting a file on a remote file system and virtualizing it as a block device.
- Performance differences between iSCSI and NFS are normally negligible in virtualized environments. For a detailed investigation, refer to NetApp TR-3808: VMware vSphere and ESX 3.5 Multiprotocol Performance Comparison Using FC, iSCSI, and NFS.

Fibre Channel

The following guidelines apply to using FC:

- The current maximum number of FC LUNs per NetApp cluster is either 8,192 or 49,152, depending on the FAS model number. (For detailed information about a particular model, refer to the NetApp Hardware Universe.) To increase this number for an OpenStack deployment, you can configure Cinder to operate with multiple NetApp clusters through multi-backend support.
- LUNs consume more management resources, and some management tools also have limitations on the number of LUNs.

Storage Service Catalog

The Storage Service Catalog concept describes a set of capabilities that enable efficient, repeated, and consistent use and management of storage resources by the definition of policy-based services and the mapping of those services to the backend storage technology. From the detailed technical implementations of the features at a storage backend, it abstracts a set of simplified configuration options.

The storage features are organized into groups based on the customer's needs to achieve a particular scenario or use case. Based on the catalog of the storage features, intelligent provisioning decisions are made by enabling the storage service catalog through infrastructure or software. In OpenStack, this is achieved by using both the Cinder filter scheduler and the NetApp driver, combining extra-specs support for volume type together with the filter scheduler. Prominent features that are used in the NetApp driver include mirroring, deduplication, compression, and thin provisioning.

When the NetApp unified driver is used with a clustered Data ONTAP storage system, you can leverage extra specs with Cinder volume types so that Cinder volumes are created on storage backends that have certain properties configured (for example, QoS, mirroring, or compression).

Extra specs are associated with Cinder volume types, so when users request volumes of a particular volume type, they are created on storage backends that meet the list of requirements (such as available space or extra specs). For more information on the list of supported extra specs, refer to the documentation available at NetApp Supported Extra Specs for Clustered Data ONTAP.

OnCommand Workflow Automation

The NetApp Cinder driver can operate in two independent modes:

- **Direct mode**, in which Cinder processes interact directly with NetApp FAS storage systems
- **Intermediated mode**, in which Cinder processes interact with an additional software entity that issues provisioning and management requests on behalf of Cinder

NetApp OnCommand WFA is a flexible framework that provides automation for storage-related tasks, customization, scripting capabilities, and integration with higher-order IT systems such as orchestration software through web services.

Although WFA can be used with the NetApp unified Cinder driver, deploying both Cinder and WFA introduces additional complexity, management entities, and potential points of failure in a cloud architecture. If you have an existing set of workflows that are written in the WFA framework and you want to leverage them in place of the default provisioning behavior of the Cinder driver operating directly against the Data ONTAP system, then it might be desirable to use the intermediated mode.

Best Practice
Unless you have a significant existing investment in OnCommand WFA that you want to leverage in an OpenStack deployment, NetApp recommends starting with the direct mode of operation when deploying Cinder with a NetApp Data ONTAP cluster.

6.5 Concept Map for E-Series and Cinder

This section describes how E-Series disk pools work with Cinder backends and explains both Cinder and E-Series volumes.

Cinder Backends and E-Series Disk Pools

An E-Series disk pool is a collection of 11 or more drives in a storage array that have the same spindle speed, the same security level, and preferably the same capacity to make the most efficient use of the drives.

A storage array can contain one or more disk pools, although the benefits of using a disk pool increase as the number of drives in a disk pool increases. Creating a disk pool with the largest number of similar drives is the preferred approach. However, if not all drives in the storage array have the same characteristics, or if you want certain drives to support different applications, you can create more than one disk pool on your storage array. In most cases, there is no practical limit on the number of drives in a disk pool, although a disk pool cannot contain more drives than the maximum limit for each storage array.

The NetApp Cinder driver for E-Series can be configured so that a Cinder backend is able to limit provisioning to only a specified subset of disk pools. This approach can be used to segment Cinder backends for a variety of reasons, including tenancy and media type.

Important

The NetApp Cinder driver for E-Series does not support interaction with traditional RAID with volume groups defined in the storage array.

Cinder Volumes and E-Series Volumes

A volume is a logical component that a host uses to organize data storage on a storage array. The host OS sees a volume as a single drive even though data is written across several physical drives in a disk pool or a volume group. Each volume is mapped to a logical unit number (LUN) that a host uses to access a volume.

Important

When using the NetApp Cinder driver for E-Series, Cinder volumes are represented in the E-Series storage array as E-Series volumes (LUNs).

6.6 Storage Considerations for E-Series

This section describes considerations related to using the SANtricity Web Services Proxy plug-in.

SANtricity Web Services Proxy

The NetApp SANtricity Web Services Proxy provides access through standard HTTP/HTTPS mechanisms to configure management services for E-Series storage arrays. You can install Web Services Proxy on either a Linux or a Windows OS. Because Web Services Proxy satisfies the client request by collecting data or executing configuration change requests to a target storage array, the Web Services Proxy module issues SYMbol requests to the target storage arrays. Web Services Proxy provides a REST-style API for managing E-Series controllers. The API enables the integration of storage array management into OpenStack.

Best Practice

Currently, using Cinder with a NetApp E-Series system requires the use of the SANtricity Web Services Proxy in the intermediated mode.

The Web Services Proxy can be deployed in an active-passive topology. The content of `arraydata.xml` must be manually replicated between instances of the proxy if the failover is to be undetectable from the Cinder driver. If the contents of the `arraydata.xml` files are out of sync, the `cinder-volume` process must be restarted in order to continue processing provisioning or management requests.

7 OpenStack Shared File System Service (Manila)

The OpenStack Shared File System (Manila) service provides management for persistent shared file system resources. The Shared File System service was originally conceived of as an extension to the Block Storage service (Cinder), but it emerged as an official, independent project in the Grizzly release. Manila became a formally incubated OpenStack project in the Juno release, and it became generally ready for production deployment in the Kilo release.

Manila is typically deployed with other OpenStack services, such as Compute, Object Storage, or Image, as part of a larger, more comprehensive cloud infrastructure. This is not an explicit requirement, because Manila has been successfully deployed as a standalone solution for shared file system provisioning and lifecycle management. As a management service, Manila controls the provisioning and lifecycle

management of shared file systems. It does not reside in the I/O (data) path between clients and the storage controller hosting the Manila shares. Access rules can be configured by Manila clients.

A Manila share is the fundamental resource unit allocated by the service. It represents an allocation of a persistent, readable, and writable shared file system that can be accessed by OpenStack compute instances or by clients outside OpenStack.

Figure 25 shows the logical architecture of Manila. As a management service, Manila controls the provisioning and lifecycle management of the Shared File System service. It does not reside in the I/O (data) path between the hypervisor and the storage controller.

Figure 25) Logical architecture of Manila.

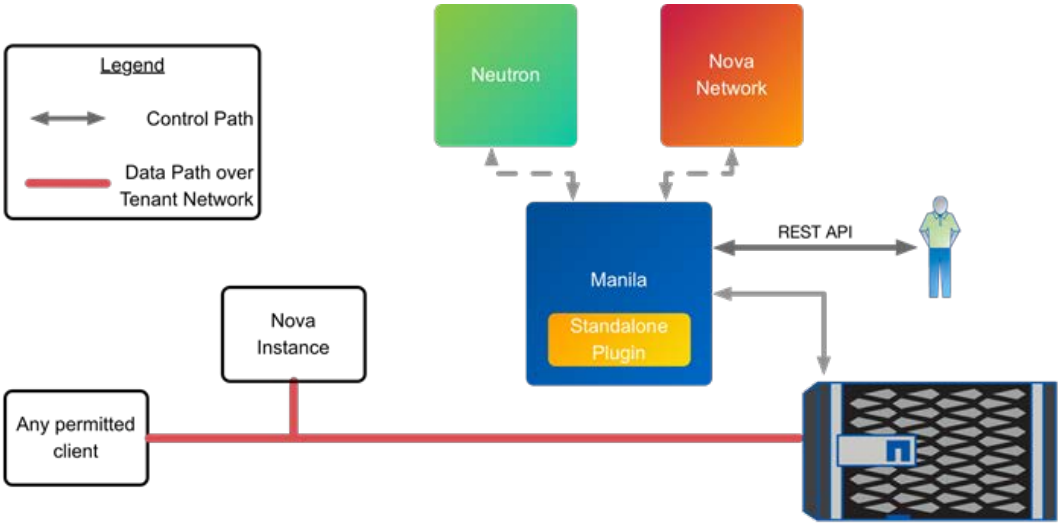


Table 6 describes the four processes that make up the Manila service.

Table 6) Overview of Manila processes.

Process Name	HA Strategy	Location	Description
`manila-api`	Active-active	Controller node	A WSGI application that accepts and validates REST (JSON or XML) requests from clients and routes the request to other Manila processes
`manila-scheduler`	Active-active	Controller node	Determines which backend should serve as the destination for a share creation request **Note:** It maintains nonpersistent state for backends (for example, available capacity, capabilities, and supported extra specs) that can be leveraged when making placement decisions. The algorithm used by the scheduler can be changed through Manila configuration.
`manila-share`	Active-active	Controller node	Accepts requests from other Manila processes and serves as the operational container for Manila drivers **Note:** This process is multithreaded and typically has one thread of execution per Manila backend as defined in the Manila configuration file.

7.1 High Availability of Manila Processes

As with the other OpenStack services, all of the Manila processes communicate with each other over the messaging subsystem described in section 4.2, “Messaging Subsystem,” and persist their state in a SQL database, as described in section 4.1, “Database Subsystem.” Therefore, the Manila processes are essentially stateless and can be scaled out across multiple controller nodes to provide a highly available shared file system service.

7.2 Concept Map for Clustered Data ONTAP and Manila

The information in this section maps concepts between clustered Data ONTAP and Manila.

Manila Share Servers and Clustered Data ONTAP SVMs

Share servers are objects defined by Manila that manage the relationship between share networks and shares. Manila offers the capability for shares to be accessible through tenant-defined networks (defined in Neutron, in Nova Networking, or externally). This capability is achieved by defining a share network object, which provides the relationship to the external network and the subnet from which an IP address should be allocated as well as configured on the backend storage.

In the context of clustered Data ONTAP, a Manila share server is defined as a storage virtual machine (SVM). SVMs contain data volumes and one or more LIFs through which they serve data to clients. SVMs can contain either one or more of the data volumes known as FlexVol volumes or a single data volume that is based on an Infinite Volume. SVMs securely isolate the shared virtualized data storage and network, and each SVM appears to clients as a single dedicated SVM. Each SVM has a separate administrator authentication domain and can be managed independently by its SVM administrator.

The clustered Data ONTAP Manila driver has two operating modes for dealing with share servers:

- One that supports the dynamic management of share servers for each unique share network. This is referred to as the NetApp clustered Data ONTAP driver with share server management.
- One that supports the reuse of a single share server (SVM) for all shares hosted from a backend. This is called a NetApp clustered Data ONTAP driver without share server management.

Important

A Manila backend defines which NetApp storage cluster to use based on the configuration file:

- If data and network isolation is an important requirement for your Manila deployment, NetApp highly recommends using the NetApp clustered Data ONTAP driver with share server management. This choice requires the backend to be configured to use one of the network plug-ins.
- If you want to have a single SVM for all Manila shares, you must use the NetApp clustered Data ONTAP driver without share server management. When this driver is used, the SVM, data LIFs, and so forth must be configured outside the scope of OpenStack before the Manila driver can be used.

Manila Storage Pools

With the Kilo release of OpenStack, Manila introduced the concept of storage pools. The Manila backend storage might present one or more logical storage resource pools that Manila selects as a storage location when provisioning shares. For NetApp clustered Data ONTAP Manila drivers, a storage pool is an aggregate defined in Data ONTAP.

Manila Network Plug-Ins

A set of network plug-ins for Manila provides a variety of approaches to integrating with the network services available with OpenStack. The share manager associated with the backend calls out to the network service associated with the backend. It calls either the standalone plug-in, Neutron, or Nova

Networking to get the required networking information, such as the share IP address, the network segmentation ID, and so forth. The network plug-in that will be leveraged with a backend is configured in the Manila configuration file. Network plug-ins provide a way for the administrator to tell Manila how to manage or use the network resources. These plug-ins can be used only with the NetApp clustered Data ONTAP driver with share server management.

The following network plug-ins for Manila are valid for the Kilo release:

- **Standalone network plug-in.** IP settings (address range, subnet mask, gateway, version) are all defined through configuration options in the driver-specific stanza of the `manila.conf` file. These settings allow the administrator to tell Manila about the existing network, and they can allow the driver to create share servers without access to or dependency on network switches or routers.
- **Nova network plug-in.** When Manila end users define share networks, this plug-in allows them to create share networks that map to different Nova networks. When a new share server is created, values for segmentation protocol, IP address, netmask, protocol, and gateway are obtained from Nova network. Default values for network ID and subnet ID can be specified through configuration options in the driver-specific stanza. Values specified by end users during share networks definition take precedence over values declared in configuration.
- **Neutron network plug-in.** Use Neutron networks and subnets when defining share networks. Values for segmentation protocol, IP address, netmask, protocol, and gateway are obtained from Neutron when a new share server is created. Default values for network ID and subnet ID can be specified through configuration options in the driver-specific stanza. Values specified by end users during share networks definition take precedence over values declared in configuration.

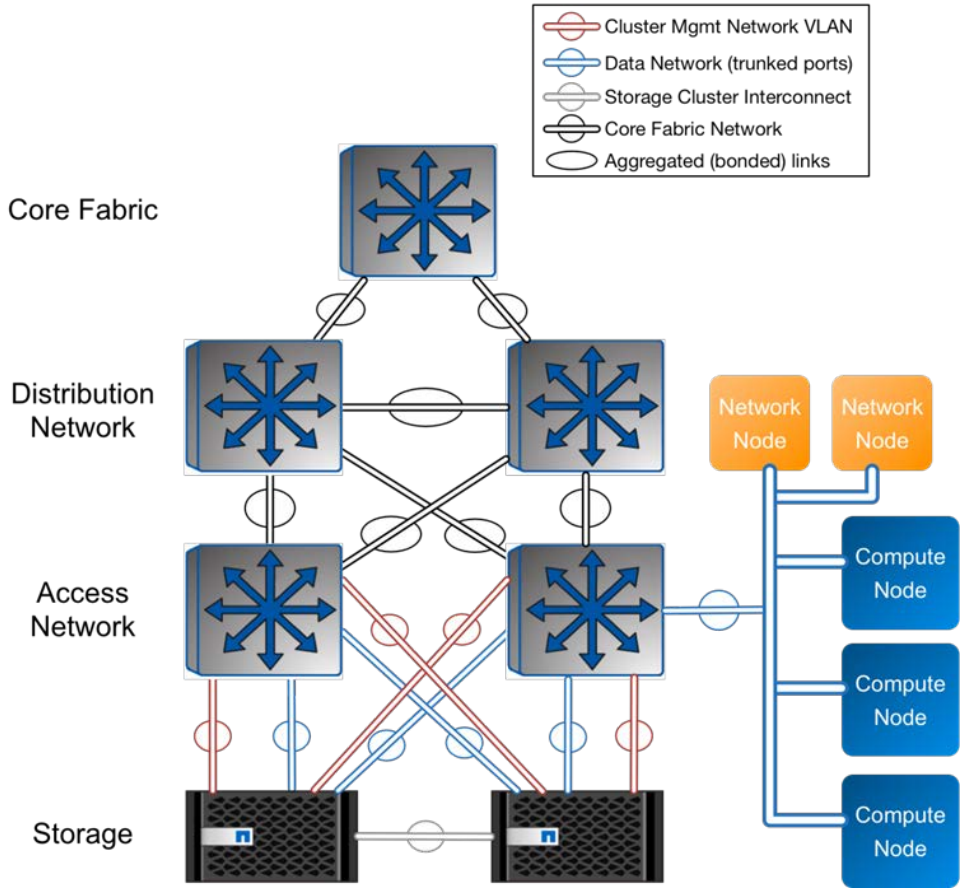
Manila Logical Network Layout

The network layout for Manila depends on the deployment choice of whether or not the Manila clustered Data ONTAP driver will manage share servers:

- **Without share server management,** the driver reuses existing data LIFs that are already configured on the cluster, and it assumes that the addresses assigned to those LIFs are routable from any clients. The driver does not make any changes to LIF configuration, and it manages export policies only to control access for specified clients.
- **With share server management,** the driver creates new SVMs for each tenant-specified share network to be associated with shares. In order for clients on the share network to be able to access the shares, the Manila driver creates new ports (with the correct VLAN tag) and new data LIFs on those ports. It queries the configured network service (through the network plug-in model previously described) to determine the correct IP address, subnet mask, and default gateway to assign to the data LIF. In this mode, it is important to have connected the physical ports of the cluster that will be used for data traffic to a switching infrastructure that is operating in trunk mode; that is, it is passing on packets with their existing VLAN tags unaltered.

Figure 26 shows the logical network layout for Manila deployments with clustered Data ONTAP with share server management enabled. The blue network represents the physical connection of data traffic between the access fabric and the Neutron network that provides connectivity to instances running on compute nodes as well as to other provider networks through the network nodes.

Figure 26) Logical network architecture of Manila with share server management.



Manila Shares and NetApp FlexVol Volumes

Data ONTAP FlexVol volumes (referred to as volumes) and OpenStack File Share Storage shares (referred to as Manila shares) are analogous. A Manila share is represented in Data ONTAP as a FlexVol volume. A FlexVol volume is a container of logical data elements such as files, Snapshot copies, clones, and so forth that is abstracted from physical elements such as disks and RAID groups.

The Manila shares can be identified uniquely by the universally unique identifier (UUID) assigned by the Manila service at the time of creation. This UUID is also part of the NetApp FlexVol volume name in the Data ONTAP cluster for easily identifying Manila shares in NetApp management software.

The connection between the consumer of the share and the Manila service providing the share can be supported with either NFS or CIFS. The list of protocols supported depends on the Manila driver deployed, the configuration of the Manila service, and the capabilities of the storage system that is connected to Manila (that is, licensing).

Manila Snapshots and NetApp Snapshot Copies

A NetApp Snapshot copy is a point-in-time file system image. Low-overhead NetApp Snapshot copies are made possible by the unique features of the WAFL storage virtualization technology that is part of Data ONTAP. The high performance of the NetApp Snapshot copy makes it highly scalable. A NetApp Snapshot copy takes only a few seconds to create, typically less than one second regardless of the size of the volume or the level of activity on the NetApp storage system. After a Snapshot copy is created, changes to data objects are reflected in updates to the current version of the objects, as though NetApp Snapshot copies did not exist. Meanwhile, the NetApp Snapshot version of the data remains completely

stable. A NetApp Snapshot copy incurs no performance overhead; users can comfortably store up to 255 NetApp Snapshot copies per FlexVol volume, all of which are accessible as read-only and online versions of the data.

Important

NetApp Snapshot copies are taken at the FlexVol level. Therefore, they can be directly leveraged in an OpenStack context when a user of Manila requests that a snapshot be taken of a particular Manila share (a FlexVol volume).

Creating Manila Shares from Manila Snapshots

The Manila service provides the capability of creating a share from a Manila snapshot. Manila leverages both NetApp Snapshot copies and FlexClone technology to efficiently create new shares based on a Manila snapshot.

NetApp FlexClone volumes are present in the same NetApp aggregate as the parent FlexVol volume. The FlexClone volume shares blocks in the parent FlexVol volume, just as all FlexClone entities and their parents share the same underlying physical data blocks, minimizing physical disk space usage.

7.3 Storage Considerations for Clustered Data ONTAP

This section explains storage considerations related to clustered Data ONTAP, including the storage protocol, the storage service catalog, and OnCommand WFA.

Storage Protocol

When clustered Data ONTAP is deployed with Manila, both NFS and CIFS are valid choices for the storage protocol. They both support Manila features, and they support Manila snapshots and cloning as well as other storage efficiency, data protection, and HA features.

NFS

The following guidelines apply for NFS:

- All of the best practices from NetApp TR-4067: Clustered Data ONTAP NFS Best Practices and Implementation Guide apply directly to Manila deployments.
- A Manila share is analogous to a NetApp FlexVol volume. To determine the maximum number of FlexVol volumes per controller, refer to the NetApp Hardware Universe for your platform limits.
- The maximum number of files in a single FlexVol volume exported through NFS depends on the size of the FlexVol volume; a 1TB FlexVol volume can have 33,554,432 files (assuming 32,000 inodes). The theoretical maximum is roughly two billion files.
- Manila supports the use of different versions of the NFS protocol. The NFS client must be able to support the version of NFS used for the Manila share.
- When Manila is used without share server management, the preconfigured SVM should have the required versions of NFS enabled.

CIFS

The following guidelines apply for CIFS:

- Follow the procedures in NetApp TR-3967: Deployment and Best Practices for Clustered Data ONTAP Windows File Services to properly configure the clustered Data ONTAP CIFS servers.
- Access rules that refer to Windows user IDs (SID) can be enforced by a clustered Data ONTAP storage system.
- If AD is used for the security services, Manila requires administration access.

- NetApp recommends using SMB version 3.
- Manila supports security services such as Kerberos, LDAP, and AD.
- When Manila is configured without share server management, the preconfigured SVM should have the required settings for CIFS/SMB enabled.

Storage Service Catalog

The Storage Service Catalog concept describes a set of capabilities that enable efficient, repeated, and consistent use and management of storage resources by the definition of policy-based services and the mapping of those services to the backend storage technology. From the detailed technical implementations of the features at a storage backend, it abstracts a set of simplified configuration options.

The storage features are organized into groups based on the customer's needs to achieve a particular scenario or use case. Based on the catalog of the storage features, intelligent provisioning decisions are made by enabling the storage service catalog through infrastructure or software. In OpenStack, this is achieved by using both the Manila filter scheduler and the NetApp driver, combining extra-specs support for share type together with the filter scheduler. Prominent features of the NetApp driver include thin provisioning, Snapshot policy, disk type, and maximum files allowed for a share.

When the NetApp unified driver is used with a clustered Data ONTAP storage system, you can leverage extra specs with Manila share types so that Manila shares are created on storage backends that have certain properties configured or apply certain attributes to the share.

Extra specs are associated with Manila share types, so that when users request shares of a particular share type, those shares are created on storage backends that meet the list of requirements (such as available space or extra specs). For more information on the list of supported extra specs, refer to the documentation available in the NetApp OpenStack Deployment and Operations Guide.

OnCommand Workflow Automation

The NetApp Manila driver can operate in two independent modes:

- **Direct mode**, in which Manila processes interact directly with NetApp FAS storage systems
- **Intermediated mode**, in which Manila processes interact with an additional software entity that issues provisioning and management requests on behalf of Cinder

NetApp OnCommand WFA is a flexible framework that provides automation for storage-related tasks, customization, scripting capabilities, and integration with higher-order IT systems such as orchestration software through web services.

Although WFA can be used with the NetApp unified Manila driver, deploying both Manila and WFA introduces additional complexity, management entities, and potential points of failure in a cloud architecture. If you have an existing set of workflows that are written in the WFA framework and you want to leverage them in place of the default provisioning behavior of the Manila driver operating directly against the Data ONTAP system, then it might be desirable to use the intermediated mode.

Best Practice

Unless you have a significant existing investment in OnCommand WFA that you want to leverage in an OpenStack deployment, NetApp recommends starting with the direct mode of operation when deploying Manila with a NetApp Data ONTAP cluster.

8 OpenStack Image Service (Glance)

The OpenStack Image Service (also known as Glance) offers discovery, registration, and delivery services for disk and server images. It provides the ability to copy a server image and immediately store it away. When multiple servers are being provisioned, a stored image can be used as a template to get new instances up and running more quickly and consistently than by installing a server OS and individually configuring additional services. It can also be used to store and catalog backups of images.

Glance can store disk and server images in a variety of image stores, including file systems (including NFS) and Object Storage. The Glance API provides a standard REST interface for querying information about disk images and allows clients to upload new images or download existing ones.

Table 7 describes the two processes that make up the Glance service.

Table 7) Overview of Glance processes.

Process Name	HA Strategy	Location	Description
`glance-api`	Active-active	Controller node	A WSGI application that accepts and validates REST (JSON or XML) requests from clients regarding images
`glance-registry`	Active-active	Controller node	Responsible for interacting with SQL database to retrieve or store image metadata

8.1 High Availability for Glance

Both Glance processes use the AMQP protocol to communicate with one another over the messaging subsystem. All stateful data managed by the Glance service is stored in its backing SQL database or in the image stores, so each Glance process is essentially stateless and can be scaled out across multiple controller nodes to provide a highly available Glance service.

8.2 Image Store

Glance stores the contents of image objects in the image store. Several implementations of the image store can be selected through the Glance configuration files, including a POSIX-compliant file system (`file`) and an object storage service (including both S3 [`s3`] and Swift [`swift`]).

The `file` store refers to a directory on a local file system where the `glance-api` service is running. The directory might refer either to locally attached storage or to a remote shared file system such as NFS.

Using Clustered Data ONTAP with File Store

The most common way to use clustered Data ONTAP with Glance is to create an NFS export from a FlexVol volume and mount the export to a directory on the local file system where the Glance services run.

Best Practice
Because there is a high probability of duplicate blocks in a repository of VM images, NetApp highly recommends enabling deduplication on the FlexVol volume where the images are stored.

Enhanced Instance Creation

In an earlier OpenStack release, NetApp contributed a capability to enhance instance creation that focuses on booting tenant-requested VM instances by OpenStack Compute Service (Nova) using persistent disk images in the shortest possible time and in the most storage capacity–efficient manner possible. This Enhanced Persistent Instance Creation feature (sometimes referred to as Rapid Cloning) is achieved by leveraging NetApp FlexClone technology as well as the NetApp Copy Offload tool. The Enhanced Instance Creation feature can significantly decrease the time elapsed when the Nova service is fulfilling image provisioning and boot requests.

When Cinder is configured to use the NFS storage protocol (refer to Section 6.4), it creates an image cache to store images that have been retrieved from a Glance image store. The image cache is always consulted during the provisioning of new Cinder volumes whose source content is an image from Glance. That is because a copy can be made efficiently by using FlexClone instead of downloading the image again from Glance.

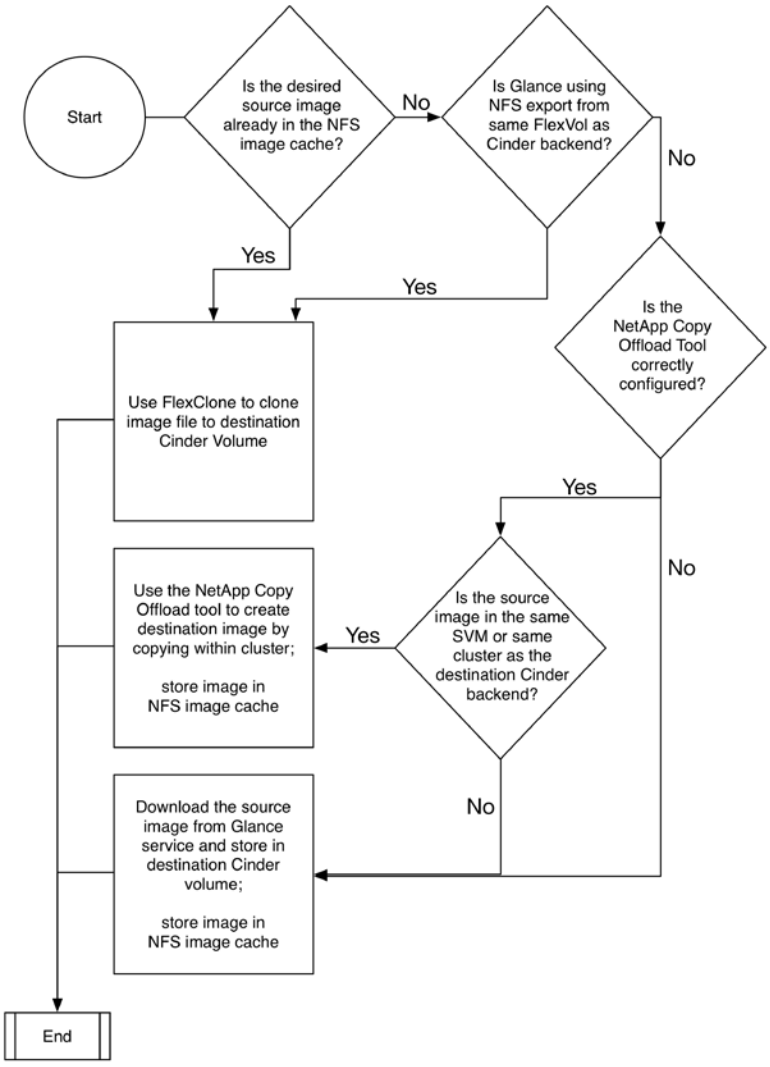
The NetApp Copy Offload tool was added in the Icehouse release to enable Glance images to be copied efficiently to a destination Cinder volume. When Cinder and Glance are configured to use the NetApp NFS Copy Offload tool, the driver attempts a controller-side copy using the NetApp NFS Copy Offload tool before reverting to downloading the image from Glance. This improves image provisioning times while reducing the consumption of bandwidth and CPU cycles on the host(s) running Glance and Cinder. Elapsed time and bandwidth consumption are reduced because the copy operation is performed completely within the storage cluster.

Figure 27 describes the workflow associated with the Enhanced Instance Cloning capability of the NetApp driver. As Figure 27 shows, the choice of FlexVol volume layout for use with Glance and Cinder can affect the workflow of the enhanced instance creation process. Although the optimal case is to have the Glance image store and Cinder datastore reside on the same FlexVol volume, the NFS image cache and NetApp Copy Offload tool can still provide the vast majority of the benefits of the rapid cloning capability without the restriction on FlexVol volume layout.

Best Practice

NetApp strongly recommends using an NFS export from a clustered Data ONTAP FlexVol volume as the backing store for Glance. That is because this enables the enhanced instance creation capability that dramatically improves provisioning times and storage utilizations in an OpenStack deployment.

Figure 27) Enhanced instance creation flowchart.



E-Series Storage Systems

E-Series and EF-Series storage systems can be used alternatively as the backing store for Glance images. An E-Series volume should be created with the SANtricity tool and mapped to the Glance host. After the volume becomes visible to the host, it is formatted with a file system and mounted, and a directory structure is created on it. This directory path can be specified in the Glance configuration.

Note: This configuration choice can limit the redundancy (and, subsequently, the SLA) of the Glance service because only one host can have the iSCSI LUN attached and mounted at a time. Use of a resource manager for failover of the Glance services and management of the mount of the LUN is required when E-Series storage systems are used as the backing storage for Glance.

9 OpenStack Object Storage Service (Swift)

OpenStack Object Storage provides a fully distributed, scale-out, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving, and data retention. Object storage does not present a traditional file system, but rather a distributed storage system for static data such as VM images, photo storage, e-mail storage, backups, and archives.

The Swift API proposes an open standard for cloud storage. It can also function as an alternative endpoint for Amazon Web Services S3 and as a CDMI server through the use of add-on components.

Swift requires node-accessible media for storing object data. This media can be drives internal to the node or external storage devices such as the NetApp E-Series storage array. This section provides information that enables NetApp storage systems to be used as the backing store for Swift object storage.

Table 8 describes the four processes that make up the Swift service.

Table 8) Overview of Swift processes.

Process Name	HA Strategy	Location	Description
`swift-proxy-server`	Active-active	Proxy node	A WSGI application that accepts and validates REST (JSON or XML) requests from clients regarding images
`swift-account-server`	Active-active	Storage node	Responsible for storage, replication, and management of account data in the object store
`swift-container-server`	Active-active	Storage node	Responsible for storage, replication, and management of container data in the object store
`swift-object-server`	Active-active	Storage node	Responsible for storage, replication, and management of object data in the object store

9.1 High Availability for Swift

The `swift-proxy-server` process is stateless and therefore can be scaled as needed based on current load. The `swift-account-server` and `swift-container-server` processes store metadata information in SQLite databases on disk that are replicated (with either individual records over HTTP or the entire database through rsync) across the cluster, similar to the way objects are replicated.

Swift also has replication and auditing capabilities for confirming that the data is replicated according to the configured policy and that the integrity of accounts, containers, and objects is upheld. As more zones (and therefore more `swift-object-server` processes) are added, the Swift services scale to maintain object resiliency and integrity according to the Swift configuration.

9.2 Swift and NetApp E-Series Storage

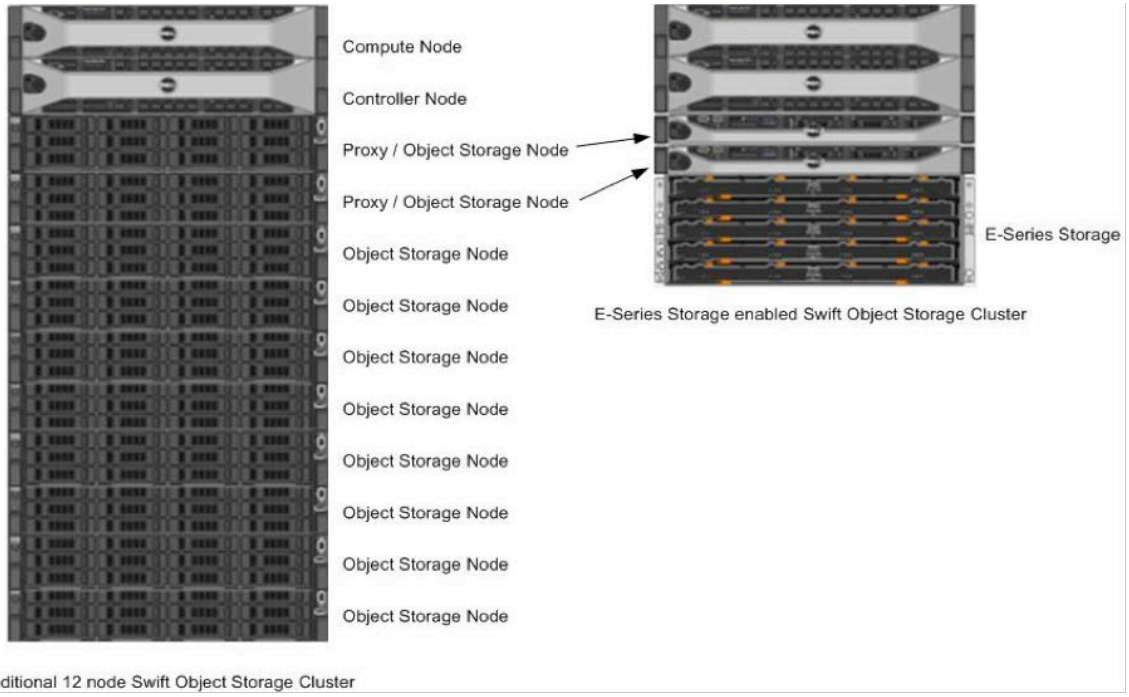
The use of E-Series storage for Swift object storage nodes offers several major advantages:

- **Reduced storage capacity and hardware requirements.** There is a dramatic reduction in the amount of storage capacity and physical hardware required for facilitating data protection through Swift's consistent hashing ring. The unique characteristics of the E-Series Dynamic Disk Pools feature enable the use of a parity protection scheme for data protection as an alternative to the default approach involving the creation of three or more copies of data. In a single site, the capacity required for object storage along with the parity overhead is an approximate 1.3 multiple of the object(s) stored. The default Swift behavior involves storing a multiple of 3.
- **Improved ability to scale.** The reduction of replicas made possible by the use of DDP has the effect of significantly reducing a typically major inhibitor to the scale that a given Swift cluster can achieve. It has been observed that the weight of replication traffic can become a limitation to scale in certain use cases.
- **Increased storage capacity efficiency.** An increase in storage capacity efficiency is associated with employing DDP.

- **Reduced Swift node hardware requirements.** Internal drive requirements for storage nodes are reduced; only OS storage is required. Disk space for Swift object data and, optionally, the OS itself is supplied by the E-Series storage array.
- **Reduced rack space, power, cooling, and footprint requirements.** Because a single storage subsystem provides storage space for multiple Swift nodes, smaller and possibly lower-power 1U nodes can be used in the cluster.

Figure 28 compares Swift footprints in traditional and E-Series storage systems.

Figure 28) Comparison of Swift footprints for traditional and E-Series storage systems.



On the left in Figure 28 is a traditional Swift cluster, which has a total storage capacity of 240TB. This requires 10 Swift object storage nodes with 12 2TB drives per system, which results in approximately 80TB of effective storage capacity, assuming that Swift uses the default replica count of 3.

Compare this traditional system to the E-Series-based cluster, shown on the right in Figure 28. The controller and compute nodes in the E-Series cluster are identical to those in the traditional system. In the E-Series cluster, the effective 160TB storage capacity of the traditional system can be obtained by using a single 4U storage subsystem. The DDP data reconstruction feature on E-Series replaces the data replication implementation of Swift. As mentioned previously, this arrangement enables a 1U server (with memory and CPU resources similar to those of the traditional cluster object nodes) to be used in the E-Series stack. This results in a 42% smaller rack footprint and approximately a 55% reduction in drive savings (that is, 120 drives versus approximately 54 drives for an E-Series–based cluster). In addition, the number of Swift object storage nodes attached to the E-Series storage array can be increased if additional object storage processing capacity is required.

DDP Reconstruction

E-Series storage can effectively serve as the storage medium for OpenStack Object Storage. The data reconstruction capabilities associated with DDP eliminate the need for data replication in zones. DDP reconstruction provides RAID 6–like data protection against multiple simultaneous drive failures in the storage subsystem. Data that resides on multiple failed drives is given top priority during reconstruction. This data, which has the highest potential for being lost if a third drive failure occurs, is reconstructed first

on the remaining optimal drives in the storage subsystem. After this critical data is reconstructed, all other data on the failed drives is reconstructed. This prioritized data reconstruction dramatically reduces the possibility of data loss from drive failure.

Swift Zones

Swift uses zoning to isolate the cluster into separate partitions in order to increase the resiliency of the cluster to failures. Swift data is replicated across the cluster in zones that are as unique as possible. A zone is an arbitrary grouping of nodes; typically, zones are established that use physical attributes of the cluster, such as geographical locations, separate networks, equipment racks, storage subsystems, or even single drives. Zoning allows the cluster to tolerate equipment failures in the cluster without data loss or loss of connectivity to the remaining clusters.

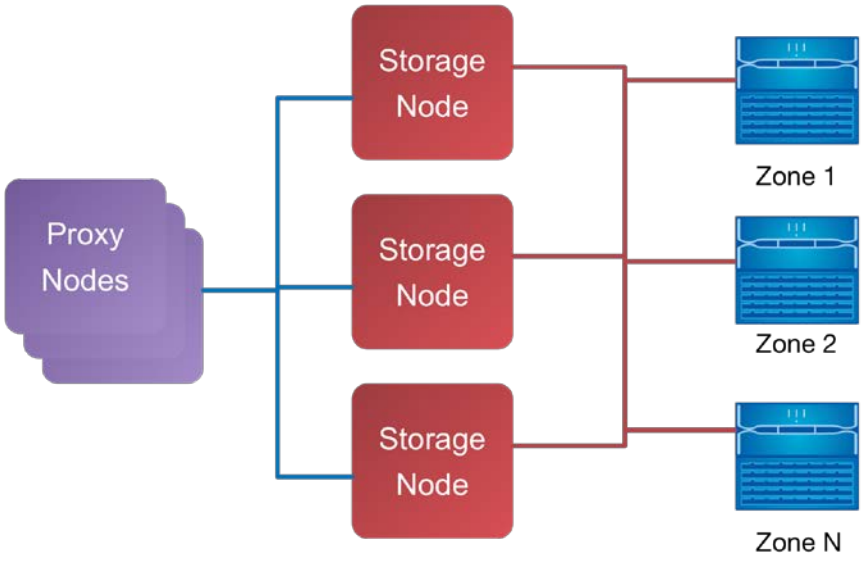
By default, Swift replicates data three times across the cluster. It replicates data across zones in a unique pattern that supports high availability and high durability for data. Swift places a replica of data in a server in an unused zone before placing it in an unused server in a zone that already has a replica of the data.

The data reconstruction feature of E-Series gives clients uninterrupted access to their data, regardless of drive failure or other component failures in the storage subsystem. When E-Series storage is used, the number of Swift data replication counts that are specified when rings are built can be reduced from three to one.

E-Series storage offers flexible configuration options that satisfy virtually all Swift zoning requirements. DDP reconstruction also eliminates the requirement of Swift data replication in a single storage array. Zoning based on E-Series storage can be defined on a storage subsystem, individual controller, or drive tray basis.

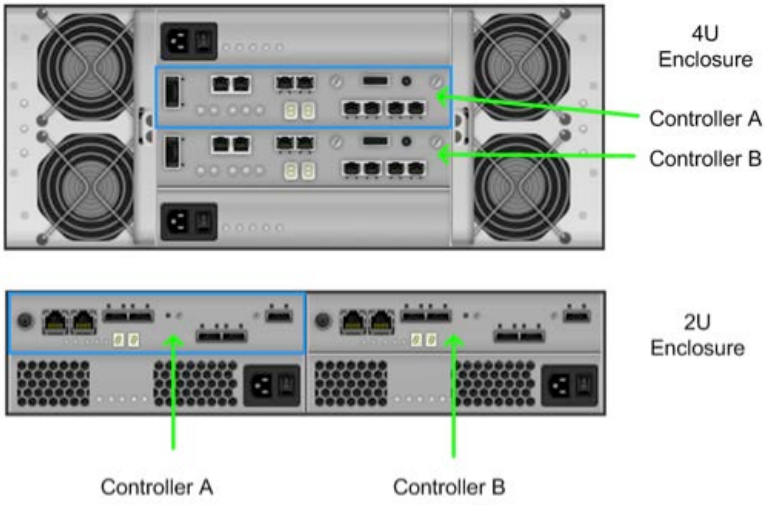
In a cluster containing several E-Series storage subsystems, one or more E-Series subsystems can be designated as a zone, as the example in Figure 29 shows.

Figure 29) Controller subsystem zoning.



E-Series storage subsystems contain two independently configurable controller modules (shown in Figure 30), which in turn communicate with drives contained in the storage subsystem and, optionally, in other externally attached E-Series drive enclosures.

Figure 30) NetApp E-Series controller layout.



For controller-based zoning, each E-Series storage controller supports two Swift object storage nodes. Each node is connected to a single controller in the storage array. LUNs for each Swift node are configured separately on drives located in the enclosure. Additional drive enclosures can also be attached to the controller module for added storage capacity.

If Swift cluster requirements require unique zones for each object node, E-Series storage arrays can effectively provide storage capacity for multiple Swift object nodes. Disk pools are created according to desired capacity. Individual drives that are designated as a disk pool are preselected by the system. Automated drive selection offers substantial benefits:

- Subsystem I/O performance is maximized.
- The impact of hardware failures in a drive enclosure or tray is minimized.
- I/O load on the subsystem is distributed as evenly as possible across all drive channels in the subsystem.

If unique zoning is not required, node connectivity is limited only by the host connectivity capacity of the E-Series storage subsystem being used.

Dynamic Disk Pools, Volumes, and Mapping

Dynamic Disk Pools should be created and sized based on the number of object storage nodes connected to the storage subsystem. A minimum of 11 drives per DDP is required, but the recommended number of drives in a DDP is equal to `N`, where `N` is the total number of drives in the storage subsystem divided by the total number of attached object storage nodes.

If no SSD drives are present, NetApp recommends creating three volumes of equal capacity on each DDP. It is important to select the Map Later option so that the mapping to a host takes place after all volumes are created. If SSDs are present, it is important to create separate DDPs that contain only SSDs. Swift documentation recommends that SSDs be leveraged for account and container-type objects.

The default mapping from SANtricity for LUN mapping is to expose all volumes to all hosts; however, this behavior is not desirable for a Swift deployment. To prevent multiple hosts from accessing the same LUN concurrently, NetApp recommends that each volume be explicitly mapped to the World Wide Name (WWN) for the appropriate host to which it should connect. If the default mapping settings are used, extreme care must be exercised to retain the correct mapping and prevent data corruption.

Swift Ring Considerations with Dynamic Disk Pools

A Swift ring represents a mapping between the names of entities stored on disk and their physical location. There are separate rings for accounts, containers, and objects. When other components must perform any operation on an object, container, or account, they must interact with the appropriate ring to determine its location in the cluster.

The ring maintains this mapping by using zones, devices, partitions, and replicas. By default, each partition in the ring is replicated three times across the cluster. The locations for a partition are stored in the mapping maintained by the ring. The ring is also responsible for determining which devices are used for handoff in failure scenarios. The number of replicas for partitions in Swift rings is set when the ring is created.

9.3 Swift and NetApp FAS Storage Systems with Clustered Data ONTAP

Swift can also be deployed in conjunction with clustered Data ONTAP and the NetApp FAS product line because iSCSI LUNs can be leveraged as block devices to provide storage for object, container, or account data. This deployment can be used when it is desirable to reuse existing Data ONTAP deployments or when the use case for object storage aligns well with the storage efficiency and data protection capabilities of clustered Data ONTAP.

Zoning Considerations with Clustered Data ONTAP

Because Swift zones are meant to segment failure domains, it is natural to think of the entire cluster deployment of clustered Data ONTAP as the architectural entity that maps to a Swift zone. Given the NetApp DataMotion™ and nondisruptive operational capabilities of clustered Data ONTAP, the cluster itself is designed to be extremely highly available and fits well with the concept of a Swift zone.

10 Other OpenStack Services

This section describes five additional OpenStack services.

10.1 OpenStack Orchestration (Heat)

OpenStack Orchestration implements a service to orchestrate multiple composite cloud applications that use the AWS CloudFormation template format through an API that is both OpenStack native and CloudFormation compatible. Heat implements an orchestration engine to launch multiple cloud applications based on those templates.

Heat templates describe the infrastructure of the cloud application, defining the necessary resources that are required for the application. It also integrates with Ceilometer to provide scaling groups as part of the templates. Heat manages the lifecycle of the application infrastructure so that when a change is made to the template, the stacks deployed from the edited template are updated.

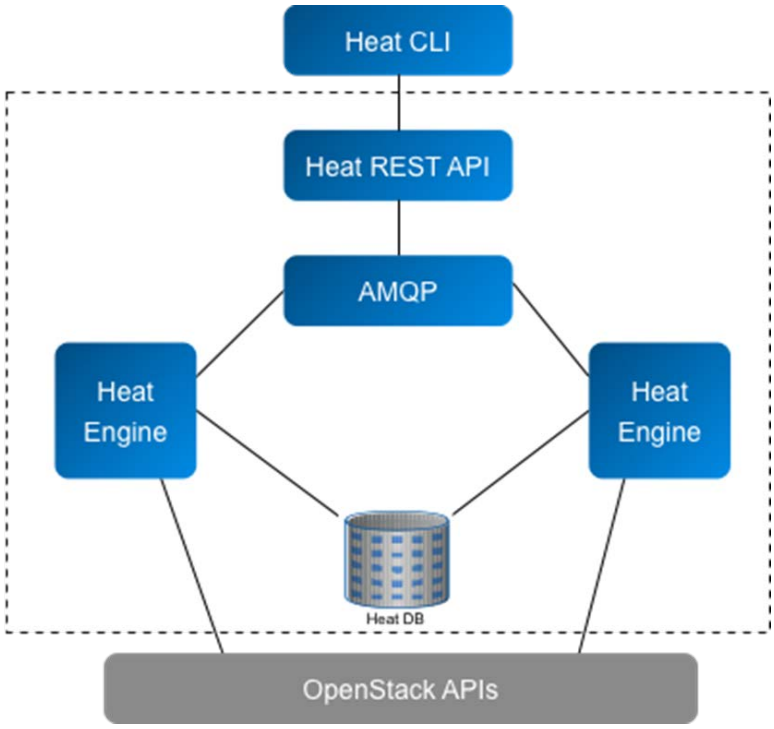
Heat primarily manages the infrastructure based on the template of the cloud application; it also integrates directly with configuration management tools such as Puppet and Chef.

Table 9 lists the processes in the Heat service. Figure 31 shows the logical architecture of Heat.

Table 9) Overview of Heat processes.

Process Name	HA Strategy	Location	Description
`heat-api`	Active-active	Controller node	Provides the REST API that processes and validates requests from clients
`heat-api-cfn`	Active-active	Controller node	Provides the implementation of the AWS CloudFormation endpoint for Heat
`heat-engine`	Active-active	Controller node	Orchestrates the launching of stacks, as well as lifecycle modifications to existing stacks

Figure 31) Heat logical architecture diagram.



High Availability for Heat

All stateful data managed by the Heat service is stored in its backing SQL database. Therefore, Heat is essentially stateless and its processes can be scaled out across multiple controller nodes to provide a highly available orchestration service.

10.2 OpenStack Identity Service (Keystone)

Keystone is an OpenStack project that provides identity, token, catalog, and policy services for use specifically in the OpenStack family. It implements the OpenStack Identity Service APIs, which provide four main types of service:

- **Identity** provides authentication and authorization management in addition to managing user, role, and tenant records and metadata.
- **Token** is used for authenticating requests after a user's credentials have been verified.
- **Catalog** provides an endpoint registry used for service discovery.
- **Policy** implements a role-based authorization engine.

Table 10 lists the single process in Keystone.

Table 10) Overview of Keystone process.

Process Name	HA Strategy	Location	Description
`keystone`	Active-active	Controller node	Provides identity, token, catalog, and policy services

High Availability for Keystone

All stateful data managed by the Keystone service is stored in its backing SQL database. Therefore, Keystone is essentially stateless and its process can be scaled out across multiple controller nodes to provide a highly available Keystone service.

Although Keystone is often configured to connect to an LDAP server or an instance of AD, this document assumes that those services have already been configured to be highly available. HA considerations for those services are outside the scope of this document.

Tokens

Two types of tokens can be generated from the Keystone service: public key infrastructure (PKI) and universally unique identifier (UUID).

PKI tokens are cryptographically encrypted by using Keystone's private key and are verified by individual services that have access to Keystone's public key, stored in an X509 certificate.

UUID tokens are randomly generated opaque strings that are generated by the Keystone service.

Important
Both PKI and UUID tokens are bearer tokens, which means that their contents must be protected from unnecessary disclosure to prevent unauthorized use and subsequent access.

PKI tokens provide a more secure and scalable solution with higher performance because each service can validate a token without querying Keystone. That is because the public key in the X509 certificate can be used to validate that a token was signed (and therefore issued) by Keystone, the holder of the private key. This approach reduces network traffic and load on Keystone.

Note: If PKI tokens are used, SSL keys and certificates must be synchronized and consistent across all instances of the Keystone process.

Backends

Keystone has a pluggable model for the datastores for each of the constituent services that it offers. It is important to use a persistent backend that refers to a highly available service, whether a database, LDAP, or AD. Although some supported backends might provide higher transactional performance (for example, in-memory key/value stores), persistence of identity information is critical to the resiliency and integrity of an OpenStack deployment.

Best Practice
NetApp recommends using PKI tokens and SQL, LDAP, or AD backends for all Keystone services.

10.3 OpenStack Telemetry (Ceilometer)

Ceilometer is an OpenStack project used for metering components in OpenStack deployments. It provides an API to query the collected data in order to provide usage data. This data is then used to provide appropriate charging and billing for the consumers of the OpenStack cloud. The data can also provide details about how the resources of an OpenStack infrastructure should be optimized.

Ceilometer has six basic components:

- **Compute agent** runs on each compute node and polls for utilization data.
- **Central agent** runs on a central server to poll for resources not tied to compute nodes.
- **Collector** runs on one or more servers to monitor the message queues for metering data.
- **Datastore** is a database capable of handling the concurrent writes of the collectors.
- **API server** is a process that runs on one or more servers to provide an API endpoint to query collected data.
- **Publishing pipeline** transforms metric data for consumption by external systems.

Table 11 lists the Ceilometer processes.

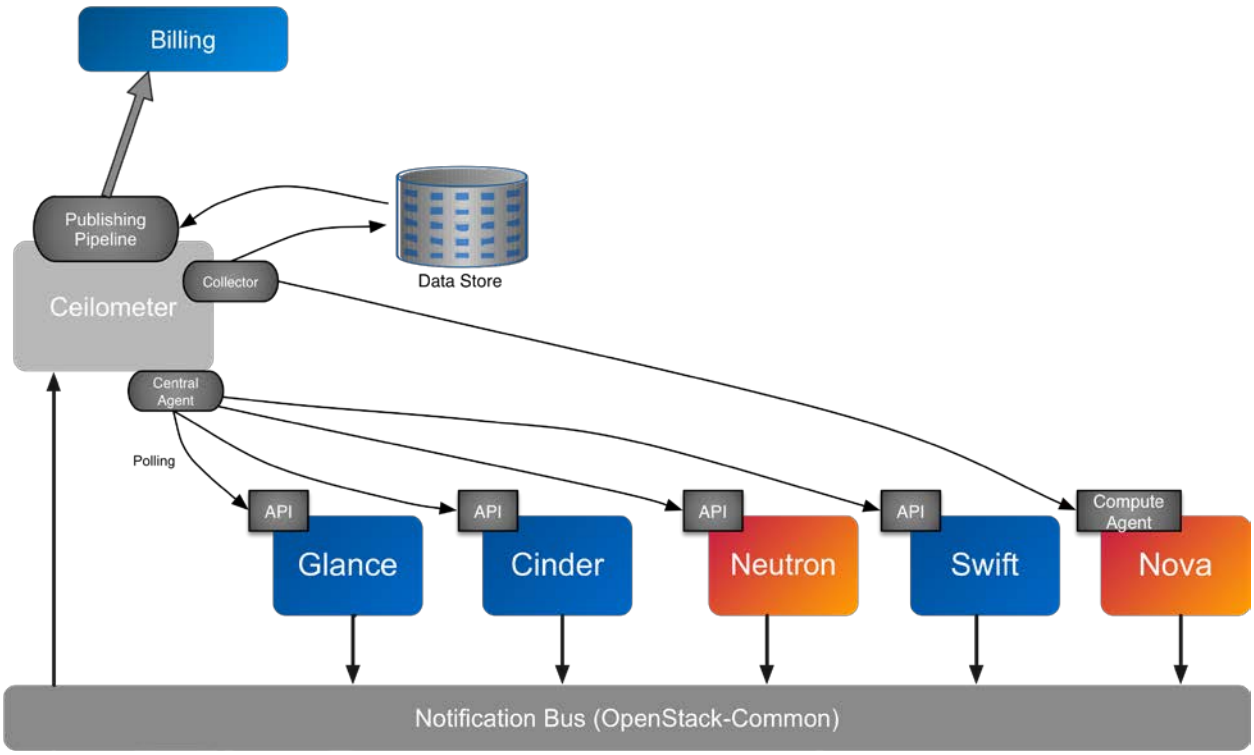
Table 11) Overview of Ceilometer processes.

Process Name	HA Strategy	Location	Description
`ceilometer-acompute`	Active-active	Compute node	Compute agent that collects metering data from compute nodes
`ceilometer-acentral`	Active-active	Controller node	Central agent that polls for resources not tied to compute nodes
`ceilometer-collector`	Active-active	Controller node	Collector agent that monitors the message bus for notifications
`ceilometer-api`	Active-active	Controller node	Provides the OpenStack-native REST API that processes and validates requests from clients
`ceilometer-alarm-evaluator`	Active-active	Controller node	Determines when an alarm should be fired based on event thresholds
`ceilometer-alarm-notifier`	Active-active	Controller node	Provides notification (alarm) when a meter reaches a threshold

The collectors deployed by Ceilometer insert data into the datastore, which is by default a MongoDB database. For HA considerations for the database used by Ceilometer, refer to Section 4.1.

Figure 32 shows the logical architecture of Ceilometer.

Figure 32) Ceilometer logical architecture.



High Availability for Ceilometer

All stateful data managed by the Ceilometer service is stored in its backing database. Therefore, the Ceilometer processes are essentially stateless and they can be scaled out across multiple controller and compute nodes to provide a highly available telemetry service.

10.4 OpenStack Dashboard (Horizon)

The OpenStack Dashboard offers administrators and users a graphical interface to access, provision, and automate cloud-based resources. The design makes it easy to plug in and use third-party products and services such as billing, monitoring, and additional management tools. The dashboard is one of several ways to interact with OpenStack resources. The dashboard provides users with a self-service portal to provision their own resources within the limits set by administrators. Horizon is a web-based user interface to OpenStack services such as Nova, Cinder, Swift, and Keystone. It is an extensible web-based application that allows cloud administrators and users to control and provision their cloud resources.

High Availability for Horizon

Horizon is essentially a stateless application, with the caveat that session information could be considered stateful. Horizon can be set up in an active-active HA configuration, with each instance of Horizon located behind a load balancer, as shown in Figure 2. In this deployment, the incoming HTTP requests are evenly distributed to all of the nodes in the Horizon deployment. However, to achieve session stickiness, it is important to direct all HTTP requests that present the same session ID to the same backend instance of Horizon, if it is available.

The use of signed cookies places the full contents of the session state into an HTTP header rather than relying on an in-memory cache or database tier to provide session state persistence.

Best Practice
NetApp recommends deploying Horizon in an active-active topology and confirming that the load balancer responsible for distributing requests is session aware. NetApp also recommends using signed cookies as a scalable way to persist state for sessions between client and servers.

10.5 OpenStack Network Service (Neutron)

The OpenStack Network Service is a pluggable, scalable, and API-driven system for managing networks and IP addresses. Like other aspects of the cloud OS, it can be used by administrators and users to increase the value of existing data center assets. Neutron prevents the network from becoming the bottleneck or a limiting factor in a cloud deployment and enables users to have self-service for their network configurations. The pluggable backend architecture lets users take advantage of basic commodity gear or advanced networking services from supported vendors. Administrators can take advantage of software-defined networking technology such as OpenFlow to allow high levels of multi-tenancy and massive scale. Neutron has an extension framework that allows additional network services, such as intrusion detection systems, load balancing, firewalls, and virtual private networks to be deployed and managed.

Neutron supports the concepts of both tenant networks and provider networks. A tenant network is a user-defined network that is completely isolated from another user's networks, even if both networks share the same physical network infrastructure. Tenant networks are supported through network namespaces, a concept present in recent Linux kernels. Neutron provider networks always map to a physical network with a gateway that exists on a router or firewall that is external to compute, controller, or network nodes. Provider network extension APIs allow a plug-in to directly control VLAN creation and trunking configuration in an external switch or router.

A reference implementation for a Neutron deployment is available from a deployment of Open vSwitch virtual switches, the `neutron-openvswitch-agent`, and the `neutron-l3-agent`. However, there are also vendor-specific plug-in agents (such as Cisco Nexus) that can both simplify the virtual switch configuration and deployment on compute nodes and directly leverage familiar physical network infrastructure to provide tenant-configurable yet fully isolated L2 and L3 networks.

Table 12 lists the processes in Neutron.

Table 12) Overview of Neutron processes.

Process Name	HA Strategy	Location	Description
`neutron-server`	Active-active	Controller node	Provides the Neutron REST API that processes and validates requests from clients
`neutron-*-agent`	Active-active	Compute node and network node	Plug-in agents provide L2 data forwarding services
`neutron-dhcp-agent`	Active-active	Network node	Manages the issuing of leases for tenant networks when IP addresses should be obtained through DHCP
`neutron-l3-agent`	Active-active	Compute node (DVR) and/or network node	Provides Layer 3 routing services in order to interconnect two or more Layer 2 networks
`neutron-metadata-agent`	Active-active	Compute node (DVR) and network node	Provides connectivity between instances and the Nova computing metadata service

Process Name	HA Strategy	Location	Description
`neutron-metering-agent`	Active-active	Network node	Provides Layer 3 traffic metering for tenant networks

High Availability for Neutron

The Neutron agents and the server process communicate through the messaging subsystem as described in Section 4.2. The neutron server persists its state in a SQL database as described in Section 4.1. The `neutron-server`, `neutron-metadata-agent`, and `neutron-metering-agent` processes are essentially stateless and can be scaled out across a number of network and controller nodes.

The `neutron-dhcp-agent` is stateless; however, it simply verifies that there is a DHCP server running and servicing tenant overlay networks. The DHCP server itself is stateful; however, multiple DHCP servers can be run on separate network nodes where the `neutron-dhcp-agent` processes are running, so that all instances of `neutron-dhcp-agent` have a DHCP server for all tenant networks. This provides the required resiliency of DHCP services from both a DHCP agent and a server level.

The `neutron-l3-agent` was made stateless in the Juno release of OpenStack and therefore is now typically deployed in an active-active topology (as long as source network address translation is not used). As of the Juno release, it is possible to run multiple copies of the `neutron-l3-agent` on different compute nodes to enable the distributed virtual routing feature.

11 Technology Requirements

This section covers the technology requirements for the OpenStack on NetApp solution.

11.1 Hardware Requirements

Table 13 lists the hardware components required to implement the solution. The hardware components used in any particular implementation of the solution might vary based on customer requirements and the set of OpenStack services that a cloud deployment would offer to its end users.

Table 13) Hardware requirements.

Hardware	Quantity
Servers	
Any server capable of running Linux and Python	Minimum of 4 (2 controller nodes, 2 compute nodes)
Storage	
NetApp FASXXXX HA pair (Model number depends on capacity and performance requirements)	Minimum of 2 nodes configured as active-active pair
Cluster Switches	
NetApp E/EF storage controllers (Model depends on capacity and performance)	2 total for cluster interconnect unless a 2-node cluster is leveraged in switchless mode
Network	
Ethernet switches (NetApp highly recommends 10GbE)	Minimum of 4 to provide redundant switching for cloud services (2 for storage networks, 2 for tenant networks)

11.2 Software Requirements

Table 14 lists the software components required to implement the solution. The software components used in any particular implementation of the solution might vary based on customer requirements.

Table 14) Software requirements.

Software	Product / Version
OpenStack	Kilo (April 2015 release)
SQL database	MySQL, PostgreSQL
NoSQL database	MongoDB
Enterprise messaging system	RabbitMQ

12 Conclusion

Most options for OpenStack integrated storage solutions aspire to offer scalability but often lack the features and performance needed for efficient and cost-effective cloud deployment at scale. NetApp platforms integrated with OpenStack offer a unique combination of advanced storage efficiency, integrated data protection, and nondisruptive operations with the capability to scale while preserving performance.

With NetApp, organizations can lower risk and enable a broad spectrum of cloud SLAs by combining the power and ingenuity of OpenStack cloud management with proven data integrity and fully developed storage provisioning, data protection, and efficiency.

Acknowledgments

The creation of this document would not have been possible without the help of several key individuals, including Jeff Applewhite, Bryan Dean, Robert Esker, Thomas Lichtenstein, and Jon Olby, all from NetApp; Steven Carter from Cisco; and Jon Benedict from Red Hat.

References

Technical Reports and Documentation

- TR-3298: RAID-DP: NetApp Implementation of Double-Parity RAID for Data Protection
  http://www.netapp.com/us/media/tr-3298.pdf
- TR-3437: Storage Subsystem Resiliency Guide
  http://www.netapp.com/us/media/tr-3437.pdf
- TR-3450: High-Availability Pair Controller Configuration Overview and Best Practices
  http://www.netapp.com/us/media/tr-3450.pdf
- TR-3601: Online MySQL Backup Using NetApp Snapshot Technology
  http://www.netapp.com/us/media/tr-3601.pdf
- TR-3657: Best Practices Guidelines for MySQL
  http://www.netapp.com/us/media/tr-3657.pdf
- TR-3802: Ethernet Storage Best Practices
  http://www.netapp.com/us/media/tr-3802.pdf

- TR-3808: VMware vSphere and ESX 3.5 Multiprotocol Performance Comparison Using FC, iSCSI, and NFS
  http://www.netapp.com/us/media/tr-3808.pdf
- TR-3848: Red Hat Enterprise Linux 6, KVM, and NetApp Storage: Best Practices Guide
  http://www.netapp.com/us/media/tr-3848.pdf
- TR-3964: Clustered Data ONTAP Security Guidance
  http://www.netapp.com/us/media/tr-3964.pdf
- TR-4067: Clustered Data ONTAP NFS Best Practice and Implementation Guide
  http://www.netapp.com/us/media/tr-4067.pdf
- TR-4068: VMware vSphere 5 on NetApp Clustered Data ONTAP: Best Practices
  http://www.netapp.com/us/media/tr-4068.pdf
- TR-4284: Reference Architecture: Deploying Red Hat Enterprise Linux OpenStack Platform 4 on NetApp Clustered Data ONTAP
  http://www.netapp.com/us/media/tr-4284.pdf
- TR-4378: Red Hat Enterprise Linux OpenStack Platform 5 on NetApp Clustered Data ONTAP
  http://www.netapp.com/us/media/tr-4378.pdf
- WP-7188: NoSQL Technologies and NetApp
  http://www.netapp.com/us/media/wp-7188.pdf
- NetApp E-Series Storage Systems Failover Drivers Guide
  https://library.netapp.com/ecm/ecm_download_file/ECMP1394845

Community Resources

- NetApp OpenStack Community Site
  http://www.netapp.com/openstack
- NetApp Hardware Universe
  http://hwu.netapp.com/
- OpenStack Documentation
  http://docs.openstack.org/
- OpenStack Security Guide
  http://docs.openstack.org/sec/

Version History

Version	Date	Document Version History
Version 2.0	May 2015	Updates for the OpenStack Kilo release (April 2015); new section about OpenStack Shared File System service (Manila)
Version 1.0	August 2014	Initial release based on the OpenStack Icehouse release (April 2014)

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.