# OpenStack Architecture Design Guide

current (2014-09-13)


To reap the benefits of OpenStack, you should plan, design, and architect your cloud properly, taking user's needs into account and understanding the use cases.

# Table of Contents

# Preface

## Conventions

The OpenStack documentation uses several typesetting conventions.

## Notices

Notices take these forms:

### Note

A handy tip or reminder.

### Important

Something you must be aware of before proceeding.

### Warning

Critical information about the risk of data loss or security issues.

## Command prompts

**$ prompt** Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

**# prompt** The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

# Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

| Revision Date | Summary of Changes |
|---|---|
| July 21, 2014 | • Initial release. |

# 1. Introduction

## Table of Contents



OpenStack is a leader in the cloud technology gold rush, as organizations of all stripes discover the increased flexibility and speed to market that self-service cloud and Infrastructure-as-a-Service (IaaS) provides. To truly reap those benefits, however, the cloud must be designed and architected properly.

A well-architected cloud provides a stable IT environment that offers easy access to needed resources, usage-based expenses, extra capacity on demand, disaster recovery, and a secure environment, but a well-architected cloud does not magically build itself. It requires careful consideration of a multitude of factors, both technical and non-technical.

There is no single architecture that is "right" for an OpenStack cloud deployment. OpenStack can be used for any number of different purposes, and each of them has its own particular requirements and architectural peculiarities.

This book is designed to look at some of the most common uses for OpenStack clouds (and even some that are less common, but provide a good example) and explain what issues need to be considered and why, along with a wealth of knowledge and advice to help an organization to design and build a well-architected OpenStack cloud that will fit its unique requirements.

## Intended audience

This book has been written for architects and designers of OpenStack clouds. This book is not intended for people who are deploying OpenStack. For a guide on deploying and operating OpenStack, please refer to the *OpenStack Operations Guide* (http://docs.openstack.org/openstack-ops).

The reader should have prior knowledge of cloud architecture and principles, experience in enterprise system design, Linux and virtualization experience, and a basic understanding of networking principles and protocols.

# How this book is organized

This book has been organized into various chapters that help define the use cases associated with making architectural choices related to an *OpenStack* cloud installation. Each chapter is intended to stand alone to encourage individual chapter readability, however each chapter is designed to contain useful information that may be applicable in situations covered by other chapters. Cloud architects may use this book as a comprehensive guide by reading all of the use cases, but it is also possible to review only the chapters which pertain to a specific use case. When choosing to read specific use cases, note that it may be necessary to read more than one section of the guide to formulate a complete design for the cloud. The use cases covered in this guide include:

- General purpose: A cloud built with common components that should address 80% of common use cases.
- Compute focused: A cloud designed to address compute intensive workloads such as high performance computing (HPC).
- Storage focused: A cloud focused on storage intensive workloads such as data analytics with parallel file systems.
- Network focused: A cloud depending on high performance and reliable networking, such as a *content delivery network (CDN)*.
- Multi-site: A cloud built with multiple sites available for application deployments for geographical, reliability or data locality reasons.
- Hybrid cloud: An architecture where multiple disparate clouds are connected either for failover, hybrid cloud bursting, or availability.
- Massively scalable: An architecture that is intended for cloud service providers or other extremely large installations.

A chapter titled Specialized cases provides information on architectures that have not previously been covered in the defined use cases.

Each chapter in the guide is then further broken down into the following sections:

- Introduction: Provides an overview of the architectural use case.
- User requirements: Defines the set of user considerations that typically come into play for that use case.
- Technical considerations: Covers the technical issues that must be accounted when dealing with this use case.
- Operational considerations: Covers the ongoing operational tasks associated with this use case and architecture.
- Architecture: Covers the overall architecture associated with the use case.
- Prescriptive examples: Presents one or more scenarios where this architecture could be deployed.

A glossary covers the terms used in the book.

# Why and how we wrote this book

The velocity at which OpenStack environments are moving from proof-of-concepts to production deployments is leading to increasing questions and issues related to architecture design considerations. By and large these considerations are not addressed in the existing documentation, which typically focuses on the specifics of deployment and configuration options or operational considerations, rather than the bigger picture.

We wrote this book to guide readers in designing an OpenStack architecture that meets the needs of their organization. This guide concentrates on identifying important design considerations for common cloud use cases and provides examples based on these design guidelines. This guide does not aim to provide explicit instructions for installing and configuring the cloud, but rather focuses on design principles as they relate to user requirements as well as technical and operational considerations. For specific guidance with installation and configuration there are a number of resources already available in the OpenStack documentation that help in that area.

This book was written in a book sprint format, which is a facilitated, rapid development production method for books. For more information, see the Book Sprints website (www.booksprints.net).

This book was written in five days during July 2014 while exhausting the M&M, Mountain Dew and healthy options supply, complete with juggling

entertainment during lunches at VMware's headquarters in Palo Alto. The event was also documented on Twitter using the #OpenStackDesign hashtag. The Book Sprint was facilitated by Faith Bosworth and Adam Hyde.

We would like to thank VMware for their generous hospitality, as well as our employers, Cisco, Cloudscaling, Comcast, EMC, Mirantis, Rackspace, Red Hat, Verizon, and VMware, for enabling us to contribute our time. We would especially like to thank Anne Gentle and Kenneth Hui for all of their shepherding and organization in making this happen.

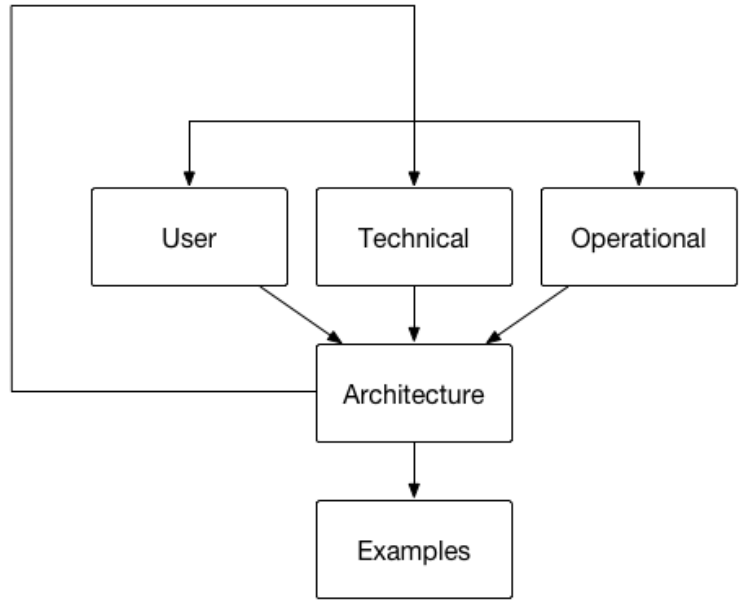
The author team includes:

- Kenneth Hui (EMC) @hui_kenneth
- Alexandra Settle (Rackspace) @dewsday
- Anthony Veiga (Comcast) @daaelar
- Beth Cohen (Verizon) @bfcohen
- Kevin Jackson (Rackspace) @itarchitectkev
- Maish Saidel-Keesing (Cisco) @maishsk
- Nick Chase (Mirantis) @NickChase
- Scott Lowe (VMware) @scott_lowe
- Sean Collins (Comcast) @sc68cal
- Sean Winn (Cloudscaling) @seanmwinn
- Sebastian Gutierrez (Red Hat) @gutseb
- Stephen Gordon (Red Hat) @xsgordon
- Vinny Valdez (Red Hat) @VinnyValdez

# Methodology

The magic of the cloud is that it can do anything. It is both robust and flexible, the best of both worlds. Yes, the cloud is highly flexible and it can do almost anything, but to get the most out of a cloud investment, it is important to define how the cloud will be used by creating and testing use cases.

This is the chapter that describes the thought process behind how to design a cloud architecture that best suits the intended use.




The diagram shows at a very abstract level the process for capturing requirements and building use cases. Once a set of use cases has been defined, it can then be used to design the cloud architecture.

Use case planning can seem counter-intuitive. After all, it takes about five minutes to sign up for a server with Amazon. Amazon does not know in advance what any given user is planning on doing with it, right? Wrong. Amazon's product management department spends plenty of time figuring out exactly what would be attractive to their typical customer and honing the service to deliver it. For the enterprise, the planning process is no different, but instead of planning for an external paying customer, for example, the use could be for internal application developers or a web portal. The following is a list of the high level objectives that need to be incorporated into the thinking about creating a use case.

Overall business objectives

- Develop clear definition of business goals and requirements
- Increase project support and engagement with business, customers and end users.

Technology

- Coordinate the OpenStack architecture across the project and leverage OpenStack community efforts more effectively.
- Architect for automation as much as possible to speed development and deployment.
- Use the appropriate tools for the development effort.
- Create better and more test metrics and test harnesses to support continuous and integrated development, test processes and automation.

Organization

- Better messaging of management support of team efforts
- Develop better cultural understanding of Open Source, cloud architectures, Agile methodologies, continuous development, test and integration, overall development concepts in general

As an example of how this works, consider a business goal of using the cloud for the company's E-commerce website. This goal means planning for applications that will support thousands of sessions per second, variable workloads, and lots of complex and changing data. By identifying the key metrics, such as number of concurrent transactions per second, size of database, and so on, it is possible to then build a method for testing the assumptions.

**Develop functional user scenarios.** Develop functional user scenarios that can be used to develop test cases that can be used to measure overall project trajectory. If the organization is not ready to commit to an application or applications that can be used to develop user requirements, it needs to create requirements to build valid test harnesses and develop usable metrics. Once the metrics are established, as requirements change, it is easier to respond to the changes quickly without having to worry overly much about setting the exact requirements in advance. Think of this as creating ways to configure the system, rather than redesigning it every time there is a requirements change.

**Limit cloud feature set.** Create requirements that address the pain points, but do not recreate the entire OpenStack tool suite. The requirement to build OpenStack, only better, is self-defeating. It is important to limit scope creep by concentrating on developing a platform that will address tool limitations for the requirements, but not recreating the entire suite of tools. Work with technical product owners to establish critical features that are needed for a successful cloud deployment.

# Application cloud readiness

Although the cloud is designed to make things easier, it is important to realize that "using cloud" is more than just firing up an instance and dropping an application on it. The "lift and shift" approach works in certain situations, but there is a fundamental difference between clouds and traditional bare-metal-based environments, or even traditional virtualized environments.

In traditional environments, with traditional enterprise applications, the applications and the servers that run on them are "pets". They're lovingly crafted and cared for, the servers have names like Gandalf or Tardis, and if they get sick, someone nurses them back to health. All of this is designed so that the application does not experience an outage.

In cloud environments, on the other hand, servers are more like cattle. There are thousands of them, they get names like NY-1138-Q, and if they get sick, they get put down and a sysadmin installs another one. Traditional applications that are unprepared for this kind of environment, naturally will suffer outages, lost data, or worse.

There are other reasons to design applications with cloud in mind. Some are defensive, such as the fact that applications cannot be certain of exactly where or on what hardware they will be launched, they need to be flexible, or at least adaptable. Others are proactive. For example, one of the advantages of using the cloud is scalability, so applications need to be designed in such a way that they can take advantage of those and other opportunities.

# Determining whether an application is cloud-ready

There are several factors to take into consideration when looking at whether an application is a good fit for the cloud.

**Structure**

A large, monolithic, single-tiered legacy application typically isn't a good fit for the cloud. Efficiencies are gained when load can be spread over several instances, so that a failure in one part of the system can be mitigated without affecting other parts of the system, or so that scaling can take place where the app needs it.

| | |
|---|---|
| **Dependencies** | Applications that depend on specific hardware—such as a particular chip set or an external device such as a fingerprint reader—might not be a good fit for the cloud, unless those dependencies are specifically addressed. Similarly, if an application depends on an operating system or set of libraries that cannot be used in the cloud, or cannot be virtualized, that is a problem. |
| **Connectivity** | Self-contained applications or those that depend on resources that are not reachable by the cloud in question, will not run. In some situations, work around these issues with custom network setup, but how well this works depends on the chosen cloud environment. |
| **Durability and resilience** | Despite the existence of SLAs, things break: servers go down, network connections are disrupted, or too many tenants on a server make a server unusable. An application must be sturdy enough to contend with these issues. |

# Designing for the cloud

Here are some guidelines to keep in mind when designing an application for the cloud:

- Be a pessimist: Assume everything fails and design backwards. Love your chaos monkey.
- Put your eggs in multiple baskets: Leverage multiple providers, geographic regions and availability zones to accommodate for local availability issues. Design for portability.
- Think efficiency: Inefficient designs will not scale. Efficient designs become cheaper as they scale. Kill off unneeded components or capacity.
- Be paranoid: Design for defense in depth and zero tolerance by building in security at every level and between every component. Trust no one.

- But not too paranoid: Not every application needs the platinum solution. Architect for different SLA's, service tiers and security levels.
- Manage the data: Data is usually the most inflexible and complex area of a cloud and cloud integration architecture. Don't short change the effort in analyzing and addressing data needs.
- Hands off: Leverage automation to increase consistency and quality and reduce response times.
- Divide and conquer: Pursue partitioning and parallel layering wherever possible. Make components as small and portable as possible. Use load balancing between layers.
- Think elasticity: Increasing resources should result in a proportional increase in performance and scalability. Decreasing resources should have the opposite effect.
- Be dynamic: Enable dynamic configuration changes such as auto scaling, failure recovery and resource discovery to adapt to changing environments, faults and workload volumes.
- Stay close: Reduce latency by moving highly interactive components and data near each other.
- Keep it loose: Loose coupling, service interfaces, separation of concerns, abstraction and well defined API's deliver flexibility.
- Be cost aware: Autoscaling, data transmission, virtual software licenses, reserved instances, and so on can rapidly increase monthly usage charges. Monitor usage closely.

# 2. General purpose

## Table of Contents



An OpenStack general purpose cloud is often considered a starting point for building a cloud deployment. General purpose clouds, by their nature, balance the components and do not emphasize (or heavily emphasize) any particular aspect of the overall computing environment. The expectation is that the compute, network, and storage components will be given equal weight in the design. General purpose clouds can be found in private, public, and hybrid environments. They lend themselves to many different use cases but, since they are homogeneous deployments, they are not suited to specialized environments or edge case situations. Common uses to consider for a general purpose cloud could be, but are not limited to, providing a simple database, a web application runtime environment, a shared application development platform, or lab test bed. In other words, any use case that would benefit from a scale-out rather than a scale-up approach is a good candidate for a general purpose cloud architecture.

A general purpose cloud, by definition, is something that is designed to have a range of potential uses or functions; not specialized for a specific use. General purpose architecture is largely considered a scenario that would address 80% of the potential use cases. The infrastructure, in itself, is a specific use case. It is also a good place to start the design process. As the most basic cloud service model, general purpose clouds are designed to be platforms suited for general purpose applications.

General purpose clouds are limited to the most basic components, but they can include additional resources such as:

- Virtual-machine disk image library
- Raw block storage
- File or object storage

- Firewalls
- Load balancers
- IP addresses
- Network overlays or virtual local area networks (VLANs)
- Software bundles

# User requirements

The general purpose cloud is built following the *Infrastructure-as-a-Service (IaaS)* model; as a platform best suited for use cases with simple requirements. The general purpose cloud user requirements themselves are typically not complex. However, it is still important to capture them even if the project has minimum business and technical requirements such as a proof of concept (PoC) or a small lab platform.

These user considerations are written from the perspective of the organization that is building the cloud, not from the perspective of the end-users who will consume cloud services provided by this design.

**Cost**

Financial factors are a primary concern for any organization. Since general purpose clouds are considered the baseline from which all other cloud architecture environments derive, cost will commonly be an important criteria. This type of cloud, however, does not always provide the most cost-effective environment for a specialized application or situation. Unless razor-thin margins and costs have been mandated as a critical factor, cost should not be the sole consideration when choosing or designing a general purpose architecture.

**Time to market**

Another common business factor in building a general purpose cloud is the ability to deliver a service or product more quickly and flexibly. In the modern hyper-fast business world, being able to deliver a product in six months instead of two years is often a major driving force behind the decision to build a general purpose cloud. General purpose clouds allow users to self-provision and gain access to com-

pute, network, and storage resources on-demand thus decreasing time to market. It may potentially make more sense to build a general purpose PoC as opposed to waiting to finalize the ultimate use case for the system. The tradeoff with taking this approach is the risk that the general purpose cloud is not optimized for the actual final workloads. The final decision on which approach to take will be dependent on the specifics of the business objectives and time frame for the project.

**Revenue opportunity**

The revenue opportunity for a given cloud will vary greatly based on the intended use case of that particular cloud. Some general purpose clouds are built for commercial customer facing products, but there are plenty of other reasons that might make the general purpose cloud the right choice. A small cloud service provider (CSP) might want to build a general purpose cloud rather than a massively scalable cloud because they do not have the deep financial resources needed, or because they do not or will not know in advance the purposes for which their customers are going to use the cloud. For some users, the advantages cloud itself offers mean an enhancement of revenue opportunity. For others, the fact that a general purpose cloud provides only baseline functionality will be a disincentive for use, leading to a potential stagnation of potential revenue opportunities.

# Legal requirements

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.

- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.
- Data compliance policies governing certain types of information need to reside in certain locations due to regular issues - and more important cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

# Technical requirements

Technical cloud architecture requirements should be weighted against the business requirements.

| | |
|---|---|
| **Performance** | As a baseline product, general purpose clouds do not provide optimized performance for any particular function. While a general purpose cloud should provide enough performance to satisfy average user considerations, performance is not a general purpose cloud customer driver. |
| **No predefined usage model** | The lack of a pre-defined usage model enables the user to run a wide variety of applications without having to know the application requirements in advance. This provides a degree of independence and flexibility that no other cloud scenarios are able to provide. |
| **On-demand and self-service application** | By definition, a cloud provides end users with the ability to self-provision computing power, storage, networks, and software in a simple and flexible way. The user must be able to scale their resources up to a substantial level without disrupting the underlying host operations. One of the benefits of using a general purpose cloud architecture is the ability to start with limited |

| | |
|---|---|
| | resources and increase them over time as the user demand grows. |
| **Public cloud** | For a company interested in building a commercial public cloud offering based on OpenStack, the general purpose architecture model might be the best choice because the designers are not going to know the purposes or workloads for which the end users will use the cloud. |
| **Internal consumption (private) cloud** | Organizations need to determine if it makes the most sense to create their own clouds internally. The main advantage of a private cloud is that it allows the organization to maintain complete control over all the architecture and the cloud components. One caution is to think about the possibility that users will want to combine using the internal cloud with access to an external cloud. If that case is likely, it might be worth exploring the possibility of taking a multi-cloud approach with regard to at least some of the architectural elements. Designs that incorporate the use of multiple clouds, such as a private cloud and a public cloud offering, are described in the "Multi-Cloud" scenario, see Chapter 6, “Multi-site” [135]. |
| **Security** | Security should be implemented according to asset, threat, and vulnerability risk assessment matrices. For cloud domains that require increased computer security, network security, or information security, general purpose cloud is not considered an appropriate choice. |

# Technical considerations

When designing a general purpose cloud, there is an implied requirement to design for all of the base services generally associated with providing Infrastructure-as-a-Service: compute, network and storage. Each of these services have different resource requirements. As a result, it is important to make design decisions relating directly to the service currently under design, while providing a balanced infrastructure that provides for all services.

When designing an OpenStack cloud as a general purpose cloud, the hardware selection process can be lengthy and involved due to the sheer mass of services which need to be designed and the unique characteristics and requirements of each service within the cloud. Hardware designs need to be generated for each type of resource pool; specifically, compute, network, and storage. In addition to the hardware designs, which affect the resource nodes themselves, there are also a number of additional hardware decisions to be made related to network architecture and facilities planning. These factors play heavily into the overall architecture of an OpenStack cloud.

## Designing compute resources

It is recommended to design compute resources as pools of resources which will be addressed on-demand. When designing compute resource pools, a number of factors impact your design decisions. For example, decisions related to processors, memory, and storage within each hypervisor are just one element of designing compute resources. In addition, it is necessary to decide whether compute resources will be provided in a single pool or in multiple pools.

To design for the best use of available resources by applications running in the cloud, it is recommended to design more than one compute resource pool. Each independent resource pool should be designed to provide service for specific flavors of instances or groupings of flavors. For the purpose of this book, "instance" refers to a virtual machines and the operating system running on the virtual machine. Designing multiple resource pools helps to ensure that, as instances are scheduled onto compute hypervisors, each independent node's resources will be allocated in a way that makes the most efficient use of available hardware. This is commonly referred to as bin packing.

Using a consistent hardware design among the nodes that are placed within a resource pool also helps support bin packing. Hardware nodes select-

ed for being a part of a compute resource pool should share a common processor, memory, and storage layout. By choosing a common hardware design, it becomes easier to deploy, support and maintain those nodes throughout their life cycle in the cloud.

OpenStack provides the ability to configure overcommit ratio—the ratio of virtual resources available for allocation to physical resources present—for both CPU and memory. The default CPU overcommit ratio is 16:1 and the default memory overcommit ratio is 1.5:1. Determine the tuning of the overcommit ratios for both of these options during the design phase, as this has a direct impact on the hardware layout of your compute nodes.

As an example, consider that a m1.small instance uses 1 vCPU, 20 GB of ephemeral storage and 2,048 MB of RAM. When designing a hardware node as a compute resource pool to service instances, take into consideration the number of processor cores available on the node as well as the required disk and memory to service instances running at capacity. For a server with 2 CPUs of 10 cores each, with hyperthreading turned on, the default CPU overcommit ratio of 16:1 would allow for 640 (2 x 10 x 2 x 16) total `m1.small` instances. By the same reasoning, using the default memory overcommit ratio of 1.5:1 you can determine that the server will need at least 853GB (640 x 2,048 MB % 1.5) of RAM. When sizing nodes for memory, it is also important to consider the additional memory required to service operating system and service needs.

Processor selection is an extremely important consideration in hardware design, especially when comparing the features and performance characteristics of different processors. Some newly released processors include features specific to virtualized compute hosts including hardware assisted virtualization and technology related to memory paging (also known as EPT shadowing). These features have a tremendous positive impact on the performance of virtual machines running in the cloud.

In addition to the impact on actual compute services, it is also important to consider the compute requirements of resource nodes within the cloud. Resource nodes refers to non-hypervisor nodes providing controller, object storage, block storage, or networking services in the cloud. The number of processor cores and threads has a direct correlation to the number of worker threads which can be run on a resource node. It is important to ensure sufficient compute capacity and memory is planned on resource nodes.

Workload profiles are unpredictable in a general purpose cloud, so it may be difficult to design for every specific use case in mind. Additional com-

pute resource pools can be added to the cloud at a later time, so this unpredictability should not be a problem. In some cases, the demand on certain instance types or flavors may not justify an individual hardware design. In either of these cases, start by providing hardware designs which will be capable of servicing the most common instance requests first, looking to add additional hardware designs to the overall architecture in the form of new hardware node designs and resource pools as they become justified at a later time.

# Designing network resources

An OpenStack cloud traditionally has multiple network segments, each of which provides access to resources within the cloud to both operators and tenants. In addition, the network services themselves also require network communication paths which should also be separated from the other networks. When designing network services for a general purpose cloud, it is recommended to plan for either a physical or logical separation of network segments which will be used by operators and tenants. It is further suggested to create an additional network segment for access to internal services such as the message bus and database used by the various cloud services. Segregating these services onto separate networks helps to protect sensitive data and also protects against unauthorized access to services.

Based on the requirements of instances being serviced in the cloud, the next design choice which will affect your design is the choice of network service which will be used to service instances in the cloud. The choice between legacy networking (nova-network), as a part of OpenStack Compute, and OpenStack Networking (neutron), has tremendous implications and will have a huge impact on the architecture and design of the cloud network infrastructure.

The legacy networking (nova-network) service is primarily a layer-2 networking service that functions in two modes. In legacy networking, the two modes differ in their use of VLANs. When using legacy networking in a flat network mode, all network hardware nodes and devices throughout the cloud are connected to a single layer-2 network segment that provides access to application data.

When the network devices in the cloud support segmentation using VLANs, legacy networking can operate in the second mode. In this design model, each tenant within the cloud is assigned a network subnet which is mapped to a VLAN on the physical network. It is especially important to remember the maximum number of 4096 VLANs which can be used

within a spanning tree domain. These limitations place hard limits on the amount of growth possible within the data center. When designing a general purpose cloud intended to support multiple tenants, it is especially recommended to use legacy networking with VLANs, and not in flat network mode.

Another consideration regarding network is the fact that legacy networking is entirely managed by the cloud operator; tenants do not have control over network resources. If tenants require the ability to manage and create network resources such as network segments and subnets, it will be necessary to install the OpenStack Networking service to provide network access to instances.

OpenStack Networking is a first class networking service that gives full control over creation of virtual network resources to tenants. This is often accomplished in the form of tunneling protocols which will establish encapsulated communication paths over existing network infrastructure in order to segment tenant traffic. These methods vary depending on the specific implementation, but some of the more common methods include tunneling over GRE, encapsulating with VXLAN, and VLAN tags.

Initially, it is suggested to design at least three network segments, the first of which will be used for access to the cloud's REST APIs by tenants and operators. This is generally referred to as a public network. In most cases, the controller nodes and swift proxies within the cloud will be the only devices necessary to connect to this network segment. In some cases, this network might also be serviced by hardware load balancers and other network devices.

The next segment is used by cloud administrators to manage hardware resources and is also used by configuration management tools when deploying software and services onto new hardware. In some cases, this network segment might also be used for internal services, including the message bus and database services, to communicate with each other. Due to the highly secure nature of this network segment, it may be desirable to secure this network from unauthorized access. This network will likely need to communicate with every hardware node within the cloud.

The last network segment is used by applications and consumers to provide access to the physical network and also for users accessing applications running within the cloud. This network is generally segregated from the one used to access the cloud APIs and is not capable of communicating directly with the hardware resources in the cloud. Compute resource nodes will need to communicate on this network segment, as will any net-

work gateway services which allow application data to access the physical network outside of the cloud.

# Designing storage resources

OpenStack has two independent storage services to consider, each with its own specific design requirements and goals. In addition to services which provide storage as their primary function, there are additional design considerations with regard to compute and controller nodes which will affect the overall cloud architecture.

# Designing OpenStack Object Storage

When designing hardware resources for OpenStack Object Storage, the primary goal is to maximize the amount of storage in each resource node while also ensuring that the cost per terabyte is kept to a minimum. This often involves utilizing servers which can hold a large number of spinning disks. Whether choosing to use 2U server form factors with directly attached storage or an external chassis that holds a larger number of drives, the main goal is to maximize the storage available in each node.

It is not recommended to invest in enterprise class drives for an OpenStack Object Storage cluster. The consistency and partition tolerance characteristics of OpenStack Object Storage will ensure that data stays up to date and survives hardware faults without the use of any specialized data replication devices.

A great benefit of OpenStack Object Storage is the ability to mix and match drives by utilizing weighting within the swift ring. When designing your swift storage cluster, it is recommended to make use of the most cost effective storage solution available at the time. Many server chassis on the market can hold 60 or more drives in 4U of rack space, therefore it is recommended to maximize the amount of storage per rack unit at the best cost per terabyte. Furthermore, the use of RAID controllers is not recommended in an object storage node.

In order to achieve this durability and availability of data stored as objects, it is important to design object storage resource pools in a way that provides the suggested availability that the service can provide. Beyond designing at the hardware node level, it is important to consider rack-level and zone-level designs to accommodate the number of replicas configured to be stored in the Object Storage service (the default number of replicas is three). Each replica of data should exist in its own availability zone with its

own power, cooling, and network resources available to service that specific zone.

Object storage nodes should be designed so that the number of requests does not hinder the performance of the cluster. The object storage service is a chatty protocol, therefore making use of multiple processors that have higher core counts will ensure the IO requests do not inundate the server.

# Designing OpenStack Block Storage

When designing OpenStack Block Storage resource nodes, it is helpful to understand the workloads and requirements that will drive the use of block storage in the cloud. In a general purpose cloud these use patterns are often unknown. It is recommended to design block storage pools so that tenants can choose the appropriate storage solution for their applications. By creating multiple storage pools of different types, in conjunction with configuring an advanced storage scheduler for the block storage service, it is possible to provide tenants with a large catalog of storage services with a variety of performance levels and redundancy options.

In addition to directly attached storage populated in servers, block storage can also take advantage of a number of enterprise storage solutions. These are addressed via a plug-in driver developed by the hardware vendor. A large number of enterprise storage plug-in drivers ship out-of-the-box with OpenStack Block Storage (and many more available via third party channels). While a general purpose cloud would likely use directly attached storage in the majority of block storage nodes, it may also be necessary to provide additional levels of service to tenants which can only be provided by enterprise class storage solutions.

The determination to use a RAID controller card in block storage nodes is impacted primarily by the redundancy and availability requirements of the application. Applications which have a higher demand on input-output per second (IOPS) will influence both the choice to use a RAID controller and the level of RAID configured on the volume. Where performance is a consideration, it is suggested to make use of higher performing RAID volumes. In contrast, where redundancy of block storage volumes is more important it is recommended to make use of a redundant RAID configuration such as RAID 5 or RAID 6. Some specialized features, such as automated replication of block storage volumes, may require the use of third-party plug-ins and enterprise block storage solutions in order to provide the high demand on storage. Furthermore, where extreme performance is a requirement it may also be necessary to make use of high speed SSD disk drives' high performing flash storage solutions.

# Software selection

The software selection process can play a large role in the architecture of a general purpose cloud. Choice of operating system, selection of OpenStack software components, choice of hypervisor and selection of supplemental software will have a large impact on the design of the cloud.

Operating system (OS) selection plays a large role in the design and architecture of a cloud. There are a number of OSes which have native support for OpenStack including Ubuntu, Red Hat Enterprise Linux (RHEL), CentOS, and SUSE Linux Enterprise Server (SLES). "Native support" in this context means that the distribution provides distribution-native packages by which to install OpenStack in their repositories. Note that "native support" is not a constraint on the choice of OS; users are free to choose just about any Linux distribution (or even Microsoft Windows) and install OpenStack directly from source (or compile their own packages). However, the reality is that many organizations will prefer to install OpenStack from distribution-supplied packages or repositories (although using the distribution vendor's OpenStack packages might be a requirement for support).

OS selection also directly influences hypervisor selection. A cloud architect who selects Ubuntu, RHEL, or SLES has some flexibility in hypervisor; KVM, Xen, and LXC are supported virtualization methods available under OpenStack Compute (nova) on these Linux distributions. A cloud architect who selects Hyper-V, on the other hand, is limited to Windows Server. Similarly, a cloud architect who selects XenServer is limited to the CentOS-based dom0 operating system provided with XenServer.

The primary factors that play into OS-hypervisor selection include:

**User requirements** The selection of OS-hypervisor combination first and foremost needs to support the user requirements.

**Support** The selected OS-hypervisor combination needs to be supported by OpenStack.

**Interoperability** The OS-hypervisor needs to be interoperable with other features and services in the OpenStack design in order to meet the user requirements.

# Hypervisor

OpenStack supports a wide variety of hypervisors, one or more of which can be used in a single cloud. These hypervisors include:

- KVM (and QEMU)
- XCP/XenServer
- vSphere (vCenter and ESXi)
- Hyper-V
- LXC
- Docker
- Bare-metal

A complete list of supported hypervisors and their capabilities can be found at https://wiki.openstack.org/wiki/HypervisorSupportMatrix.

General purpose clouds should make use of hypervisors that support the most general purpose use cases, such as KVM and Xen. More specific hypervisors should then be chosen to account for specific functionality or a supported feature requirement. In some cases, there may also be a mandated requirement to run software on a certified hypervisor including solutions from VMware, Microsoft, and Citrix.

The features offered through the OpenStack cloud platform determine the best choice of a hypervisor. As an example, for a general purpose cloud that predominantly supports a Microsoft-based migration, or is managed by staff that has a particular skill for managing certain hypervisors and operating systems, Hyper-V might be the best available choice. While the decision to use Hyper-V does not limit the ability to run alternative operating systems, be mindful of those that are deemed supported. Each different hypervisor also has their own hardware requirements which may affect the decisions around designing a general purpose cloud. For example, to utilize the live migration feature of VMware, vMotion, this requires an installation of vCenter/vSphere and the use of the ESXi hypervisor, which increases the infrastructure requirements.

In a mixed hypervisor environment, specific aggregates of compute resources, each with defined capabilities, enable workloads to utilize software and hardware specific to their particular requirements. This functionality can be exposed explicitly to the end user, or accessed through defined metadata within a particular flavor of an instance.

# OpenStack components

A general purpose OpenStack cloud design should incorporate the core OpenStack services to provide a wide range of services to end-users. The OpenStack core services recommended in a general purpose cloud are:

- OpenStack *Compute* (*nova*)
- OpenStack *Networking* (*neutron*)
- OpenStack *Image Service* (*glance*)
- OpenStack *Identity* (*keystone*)
- OpenStack *dashboard* (*horizon*)
- *Telemetry* module (*ceilometer*)

A general purpose cloud may also include OpenStack *Object Storage* (*swift*). OpenStack *Block Storage* (*cinder*) may be selected to provide persistent storage to applications and instances although, depending on the use case, this could be optional.

# Supplemental software

A general purpose OpenStack deployment consists of more than just OpenStack-specific components. A typical deployment involves services that provide supporting functionality, including databases and message queues, and may also involve software to provide high availability of the OpenStack environment. Design decisions around the underlying message queue might affect the required number of controller services, as well as the technology to provide highly resilient database functionality, such as MariaDB with Galera. In such a scenario, replication of services relies on quorum. Therefore, the underlying database nodes, for example, should consist of at least 3 nodes to account for the recovery of a failed Galera node. When increasing the number of nodes to support a feature of the software, consideration of rack space and switch port density becomes important.

Where many general purpose deployments use hardware load balancers to provide highly available API access and SSL termination, software solutions, for example HAProxy, can also be considered. It is vital to ensure that such software implementations are also made highly available. This high availability can be achieved by using software such as Keepalived or Pacemaker with Corosync. Pacemaker and Corosync can provide active-ac-

tive or active-passive highly available configuration depending on the specific service in the OpenStack environment. Using this software can affect the design as it assumes at least a 2-node controller infrastructure where one of those nodes may be running certain services in standby mode.

Memcached is a distributed memory object caching system, and Redis is a key-value store. Both are usually deployed on general purpose clouds to assist in alleviating load to the Identity service. The memcached service caches tokens, and due to its distributed nature it can help alleviate some bottlenecks to the underlying authentication system. Using memcached or Redis does not affect the overall design of your architecture as they tend to be deployed onto the infrastructure nodes providing the OpenStack services.

# Performance

Performance of an OpenStack deployment is dependent on a number of factors related to the infrastructure and controller services. The user requirements can be split into general network performance, performance of compute resources, and performance of storage systems.

# Controller infrastructure

The Controller infrastructure nodes provide management services to the end-user as well as providing services internally for the operating of the cloud. The Controllers typically run message queuing services that carry system messages between each service. Performance issues related to the message bus would lead to delays in sending that message to where it needs to go. The result of this condition would be delays in operation functions such as spinning up and deleting instances, provisioning new storage volumes and managing network resources. Such delays could adversely affect an application's ability to react to certain conditions, especially when using auto-scaling features. It is important to properly design the hardware used to run the controller infrastructure as outlined above in the Hardware Selection section.

Performance of the controller services is not just limited to processing power, but restrictions may emerge in serving concurrent users. Ensure that the APIs and Horizon services are load tested to ensure that you are able to serve your customers. Particular attention should be made to the OpenStack Identity Service (Keystone), which provides the authentication and authorization for all services, both internally to OpenStack itself and to end-users. This service can lead to a degradation of overall performance if this is not sized appropriately.

# Network performance

In a general purpose OpenStack cloud, the requirements of the network help determine its performance capabilities. For example, small deployments may employ 1 Gigabit Ethernet (GbE) networking, whereas larger installations serving multiple departments or many users would be better architected with 10 GbE networking. The performance of the running instances will be limited by these speeds. It is possible to design OpenStack environments that run a mix of networking capabilities. By utilizing the different interface speeds, the users of the OpenStack environment can choose networks that are fit for their purpose. For example, web application instances may run on a public network presented through OpenStack Networking that has 1 GbE capability, whereas the back-end database uses an OpenStack Networking network that has 10 GbE capability to replicate its data or, in some cases, the design may incorporate link aggregation for greater throughput.

Network performance can be boosted considerably by implementing hardware load balancers to provide front-end service to the cloud APIs. The hardware load balancers also perform SSL termination if that is a requirement of your environment. When implementing SSL offloading, it is important to understand the SSL offloading capabilities of the devices selected.

# Compute host

The choice of hardware specifications used in compute nodes including CPU, memory and disk type directly affects the performance of the instances. Other factors which can directly affect performance include tunable parameters within the OpenStack services, for example the overcommit ratio applied to resources. The defaults in OpenStack Compute set a 16:1 over-commit of the CPU and 1.5 over-commit of the memory. Running at such high ratios leads to an increase in "noisy-neighbor" activity. Care must be taken when sizing your Compute environment to avoid this scenario. For running general purpose OpenStack environments it is possible to keep to the defaults, but make sure to monitor your environment as usage increases.

# Storage performance

When considering performance of OpenStack Block Storage, hardware and architecture choice is important. Block Storage can use enterprise back-end systems such as NetApp or EMC, use scale out storage such as GlusterFS and Ceph, or simply use the capabilities of directly attached stor-

age in the nodes themselves. Block Storage may be deployed so that traffic traverses the host network, which could affect, and be adversely affected by, the front-side API traffic performance. As such, consider using a dedicated data storage network with dedicated interfaces on the Controller and Compute hosts.

When considering performance of OpenStack Object Storage, a number of design choices will affect performance. A user's access to the Object Storage is through the proxy services, which typically sit behind hardware load balancers. By the very nature of a highly resilient storage system, replication of the data would affect performance of the overall system. In this case, 10 GbE (or better) networking is recommended throughout the storage network architecture.

# Availability

In OpenStack, the infrastructure is integral to providing services and should always be available, especially when operating with SLAs. Ensuring network availability is accomplished by designing the network architecture so that no single point of failure exists. A consideration of the number of switches, routes and redundancies of power should be factored into core infrastructure, as well as the associated bonding of networks to provide diverse routes to your highly available switch infrastructure.

The OpenStack services themselves should be deployed across multiple servers that do not represent a single point of failure. Ensuring API availability can be achieved by placing these services behind highly available load balancers that have multiple OpenStack servers as members.

OpenStack lends itself to deployment in a highly available manner where it is expected that at least 2 servers be utilized. These can run all the services involved from the message queuing service, for example RabbitMQ or QPID, and an appropriately deployed database service such as MySQL or MariaDB. As services in the cloud are scaled out, back-end services will need to scale too. Monitoring and reporting on server utilization and response times, as well as load testing your systems, will help determine scale out decisions.

Care must be taken when deciding network functionality. Currently, OpenStack supports both the legacy networking (nova-network) system and the newer, extensible OpenStack Networking. Both have their pros and cons when it comes to providing highly available access. Legacy networking, which provides networking access maintained in the OpenStack Compute code, provides a feature that removes a single point of failure when

it comes to routing, and this feature is currently missing in OpenStack Networking. The effect of legacy networking's multi-host functionality restricts failure domains to the host running that instance.

On the other hand, when using OpenStack Networking, the OpenStack controller servers or separate Networking hosts handle routing. For a deployment that requires features available in only Networking, it is possible to remove this restriction by using third party software that helps maintain highly available L3 routes. Doing so allows for common APIs to control network hardware, or to provide complex multi-tier web applications in a secure manner. It is also possible to completely remove routing from Networking, and instead rely on hardware routing capabilities. In this case, the switching infrastructure must support L3 routing.

OpenStack Networking (neutron) and legacy networking (nova-network) both have their advantages and disadvantages. They are both valid and supported options that fit different network deployment models described in the OpenStack Operations Guide.

Ensure your deployment has adequate back-up capabilities. As an example, in a deployment that has two infrastructure controller nodes, the design should include controller availability. In the event of the loss of a single controller, cloud services will run from a single controller in the event of failure. Where the design has higher availability requirements, it is important to meet those requirements by designing the proper redundancy and availability of controller nodes.

Application design must also be factored into the capabilities of the underlying cloud infrastructure. If the compute hosts do not provide a seamless live migration capability, then it must be expected that when a compute host fails, that instance and any data local to that instance will be deleted. Conversely, when providing an expectation to users that instances have a high-level of uptime guarantees, the infrastructure must be deployed in a way that eliminates any single point of failure when a compute host disappears. This may include utilizing shared file systems on enterprise storage or OpenStack Block storage to provide a level of guarantee to match service features.

For more information on high availability in OpenStack, see the *OpenStack High Availability Guide*.

# Security

A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system. Typ-

ically they have the same authentication and authorization requirements and users.

These security domains are:

- Public
- Guest
- Management
- Data

These security domains can be mapped to an OpenStack deployment individually, or combined. For example, some deployment topologies combine both guest and data domains onto one physical network, whereas in other cases these networks are physically separated. In each case, the cloud operator should be aware of the appropriate security concerns. Security domains should be mapped out against your specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which you have no authority. This domain should always be considered untrusted.

Typically used for compute instance-to-instance traffic, the guest security domain handles compute data generated by instances on the cloud but not services that support the operation of the cloud, such as API calls. Public cloud providers and private cloud providers who do not have stringent controls on instance use or who allow unrestricted Internet access to instances should consider this domain to be untrusted. Private cloud providers may want to consider this network as internal and therefore trusted only if they have controls in place to assert that they trust instances and all their tenants.

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, user names, and passwords. In most deployments this domain is considered trusted.

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. Much of the data that cross-

es this network has high integrity and confidentiality requirements and, depending on the type of deployment, may also have strong availability requirements. The trust level of this network is heavily dependent on other deployment decisions.

When deploying OpenStack in an enterprise as a private cloud it is usually behind the firewall and within the trusted network alongside existing systems. Users of the cloud are, traditionally, employees that are bound by the security requirements set forth by the company. This tends to push most of the security domains towards a more trusted model. However, when deploying OpenStack in a public facing role, no assumptions can be made and the attack vectors significantly increase. For example, the API endpoints, along with the software behind them, become vulnerable to bad actors wanting to gain unauthorized access or prevent access to services, which could lead to loss of data, functionality, and reputation. These services must be protected against through auditing and appropriate filtering.

Consideration must be taken when managing the users of the system for both public and private clouds. The identity service allows for LDAP to be part of the authentication process. Including such systems in an OpenStack deployment may ease user management if integrating into existing systems.

It's important to understand that user authentication requests include sensitive information including user names, passwords and authentication tokens. For this reason, placing the API services behind hardware that performs SSL termination is strongly recommended.

For more information OpenStack Security, see the *OpenStack Security Guide*

# Operational considerations

Many operational factors will affect general purpose cloud design choices. In larger installations, it is not uncommon for operations staff to be tasked with maintaining cloud environments. This differs from the operations staff that is responsible for building or designing the infrastructure. It is important to include the operations function in the planning and design phases of the build out.

Service Level Agreements (SLAs) are contractual obligations that provide assurances for service availability. SLAs define levels of availability that

drive the technical design, often with penalties for not meeting the contractual obligations. The strictness of the SLA dictates the level of redundancy and resiliency in the OpenStack cloud design. Knowing when and where to implement redundancy and high availability is directly affected by expectations set by the terms of the SLA. Some of the SLA terms that will affect the design include:

- Guarantees for API availability imply multiple infrastructure services combined with highly available load balancers.
- Network uptime guarantees will affect the switch design and might require redundant switching and power.
- Network security policies requirements need to be factored in to deployments.

# Support and maintainability

OpenStack cloud management requires operations staff to be able to understand and comprehend design architecture content on some level. The level of skills and the level of separation of the operations and engineering staff are dependent on the size and purpose of the installation. A large cloud service provider or a telecom provider is more likely to be managed by a specially trained, dedicated operations organization. A smaller implementation is more likely to rely on a smaller support staff that might need to take on the combined engineering, design and operations functions.

Furthermore, maintaining OpenStack installations requires a variety of technical skills. Some of these skills may include the ability to debug Python log output to a basic level and an understanding of networking concepts.

Consider incorporating features into the architecture and design that reduce the operations burden. This is accomplished by automating some of the operations functions. In some cases it may be beneficial to use a third party management company with special expertise in managing OpenStack deployments.

# Monitoring

Like any other infrastructure deployment, OpenStack clouds need an appropriate monitoring platform to ensure any errors are caught and managed appropriately. Consider leveraging any existing monitoring system to see if it will be able to effectively monitor an OpenStack environment.

While there are many aspects that need to be monitored, specific metrics that are critically important to capture include image disk utilization, or response time to the Compute API.

# Downtime

No matter how robust the architecture is, at some point components will fail. Designing for high availability (HA) can have significant cost ramifications, therefore the resiliency of the overall system and the individual components is going to be dictated by the requirements of the SLA. Downtime planning includes creating processes and architectures that support planned (maintenance) and unplanned (system faults) downtime.

An example of an operational consideration is the recovery of a failed compute host. This might mean requiring the restoration of instances from a snapshot or respawning an instance on another available compute host. This could have consequences on the overall application design. A general purpose cloud should not need to provide an ability to migrate instances from one host to another. If the expectation is that the application will be designed to tolerate failure, additional considerations need to be made around supporting instance migration. In this scenario, extra supporting services, including shared storage attached to compute hosts, might need to be deployed.

# Capacity planning

Capacity planning for future growth is a critically important and often overlooked consideration. Capacity constraints in a general purpose cloud environment include compute and storage limits. There is a relationship between the size of the compute environment and the supporting OpenStack infrastructure controller nodes required to support it. As the size of the supporting compute environment increases, the network traffic and messages will increase which will add load to the controller or networking nodes. While no hard and fast rule exists, effective monitoring of the environment will help with capacity decisions on when to scale the back-end infrastructure as part of the scaling of the compute resources.

Adding extra compute capacity to an OpenStack cloud is a horizontally scaling process as consistently configured compute nodes automatically attach to an OpenStack cloud. Be mindful of any additional work that is needed to place the nodes into appropriate availability zones and host aggregates. Make sure to use identical or functionally compatible CPUs when adding additional compute nodes to the environment otherwise live mi-

gration features will break. Scaling out compute hosts will directly affect network and other datacenter resources so it will be necessary to add rack capacity or network switches.

Another option is to assess the average workloads and increase the number of instances that can run within the compute environment by adjusting the overcommit ratio. While only appropriate in some environments, it's important to remember that changing the CPU overcommit ratio can have a detrimental effect and cause a potential increase in noisy neighbor. The added risk of increasing the overcommit ratio is more instances will fail when a compute host fails.

Compute host components can also be upgraded to account for increases in demand; this is known as vertical scaling. Upgrading CPUs with more cores, or increasing the overall server memory, can add extra needed capacity depending on whether the running applications are more CPU intensive or memory intensive.

Insufficient disk capacity could also have a negative effect on overall performance including CPU and memory usage. Depending on the back-end architecture of the OpenStack Block Storage layer, capacity might include adding disk shelves to enterprise storage systems or installing additional block storage nodes. It may also be necessary to upgrade directly attached storage installed in compute hosts or add capacity to the shared storage to provide additional ephemeral storage to instances.

For a deeper discussion on many of these topics, refer to the *OpenStack Operations Guide*.

# Architecture

Hardware selection involves three key areas:

- Compute
- Network
- Storage

For each of these areas, the selection of hardware for a general purpose OpenStack cloud must reflect the fact that a the cloud has no pre-defined usage model. This means that there will be a wide variety of applications running on this cloud that will have varying resource usage requirements. Some applications will be RAM-intensive, some applications will be CPU-in-

tensive, while others will be storage-intensive. Therefore, choosing hardware for a general purpose OpenStack cloud must provide balanced access to all major resources.

Certain hardware form factors may be better suited for use in a general purpose OpenStack cloud because of the need for an equal or nearly equal balance of resources. Server hardware for a general purpose OpenStack architecture design must provide an equal or nearly equal balance of compute capacity (RAM and CPU), network capacity (number and speed of links), and storage capacity (gigabytes or terabytes as well as Input/Output Operations Per Second (*IOPS*).

Server hardware is evaluated around four conflicting dimensions:

| | |
|---|---|
| **Server density** | A measure of how many servers can fit into a given measure of physical space, such as a rack unit [U]. |
| **Resource capacity** | The number of CPU cores, how much RAM, or how much storage a given server will deliver. |
| **Expandability** | The number of additional resources that can be added to a server before it has reached its limit. |
| **Cost** | The relative purchase price of the hardware weighted against the level of design effort needed to build the system. |

Increasing server density means sacrificing resource capacity or expandability, however, increasing resource capacity and expandability increases cost and decreases server density. As a result, determining the best server hardware for a general purpose OpenStack architecture means understanding how choice of form factor will impact the rest of the design.

- Blade servers typically support dual-socket multi-core CPUs, which is the configuration generally considered to be the "sweet spot" for a general purpose cloud deployment. Blades also offer outstanding density. As an example, both HP BladeSystem and Dell PowerEdge M1000e support up to 16 servers in only 10 rack units. However, the blade servers themselves often have limited storage and networking capacity. Additionally, the expandability of many blade servers can be limited.

- 1U rack-mounted servers occupy only a single rack unit. Their benefits include high density, support for dual-socket multi-core CPUs, and support

for reasonable RAM amounts. This form factor offers limited storage capacity, limited network capacity, and limited expandability.

- 2U rack-mounted servers offer the expanded storage and networking capacity that 1U servers tend to lack, but with a corresponding decrease in server density (half the density offered by 1U rack-mounted servers).
- Larger rack-mounted servers, such as 4U servers, will tend to offer even greater CPU capacity, often supporting four or even eight CPU sockets. These servers often have much greater expandability so will provide the best option for upgradability. This means, however, that the servers have a much lower server density and a much greater hardware cost.
- "Sled servers" are rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure. This form factor offers increased density over typical 1U-2U rack-mounted servers but tends to suffer from limitations in the amount of storage or network capacity each individual server supports.

Given the wide selection of hardware and general user requirements, the best form factor for the server hardware supporting a general purpose OpenStack cloud is driven by outside business and cost factors. No single reference architecture will apply to all implementations; the decision must flow out of the user requirements, technical considerations, and operational considerations. Here are some of the key factors that influence the selection of server hardware:

**Instance density** Sizing is an important consideration for a general purpose OpenStack cloud. The expected or anticipated number of instances that each hypervisor can host is a common metric used in sizing the deployment. The selected server hardware needs to support the expected or anticipated instance density.

**Host density** Physical data centers have limited physical space, power, and cooling. The number of hosts (or hypervisors) that can be fitted into a given metric (rack, rack unit, or floor tile) is another important method of sizing. Floor weight is an often overlooked consideration. The data center floor must be able to support the weight of the proposed number of hosts within a rack or set of racks. These factors need to be applied as part of the host

| | |
|---|---|
| | density calculation and server hardware selection. |
| **Power density** | Data centers have a specified amount of power fed to a given rack or set of racks. Older data centers may have a power density as power as low as 20 AMPs per rack, while more recent data centers can be architected to support power densities as high as 120 AMP per rack. The selected server hardware must take power density into account. |
| **Network connectivity** | The selected server hardware must have the appropriate number of network connections, as well as the right type of network connections, in order to support the proposed architecture. Ensure that, at a minimum, there are at least two diverse network connections coming into each rack. For architectures requiring even more redundancy, it might be necessary to confirm that the network connections are from diverse telecom providers. Many data centers have that capacity available. |

The selection of certain form factors or architectures will affect the selection of server hardware. For example, if the design calls for a scale-out storage architecture (such as leveraging Ceph, Gluster, or a similar commercial solution), then the server hardware selection will need to be carefully considered to match the requirements set by the commercial solution. Ensure that the selected server hardware is configured to support enough storage capacity (or storage expandability) to match the requirements of selected scale-out storage solution. For example, if a centralized storage solution is required, such as a centralized storage array from a storage vendor that has InfiniBand or FDDI connections, the server hardware will need to have appropriate network adapters installed to be compatible with the storage array vendor's specifications.

Similarly, the network architecture will have an impact on the server hardware selection and vice versa. For example, make sure that the server is configured with enough additional network ports and expansion cards to support all of the networks required. There is variability in network expansion cards, so it is important to be aware of potential impacts or interoperability issues with other components in the architecture. This is especially

true if the architecture uses InfiniBand or another less commonly used networking protocol.

# Selecting storage hardware

The selection of storage hardware is largely determined by the proposed storage architecture. Factors that need to be incorporated into the storage architecture include:

**Cost** Storage can be a significant portion of the overall system cost that should be factored into the design decision. For an organization that is concerned with vendor support, a commercial storage solution is advisable, although it is comes with a higher price tag. If initial capital expenditure requires minimization, designing a system based on commodity hardware would apply. The trade-off is potentially higher support costs and a greater risk of incompatibility and interoperability issues.

**Performance** Storage performance, measured by observing the latency of storage I-O requests, is not a critical factor for a general purpose OpenStack cloud as overall systems performance is not a design priority.

**Scalability** The term "scalability" refers to how well the storage solution performs as it expands up to its maximum designed size. A solution that continues to perform well at maximum expansion is considered scalable. A storage solution that performs well in small configurations but has degrading performance as it expands was not designed to be not scalable. Scalability, along with expandability, is a major consideration in a general purpose OpenStack cloud. It might be difficult to predict the final intended size of the implementation because there are no established usage patterns for a general purpose cloud. Therefore, it may become necessary to expand the initial deployment in order to accommodate growth and user demand. The ability of the storage solution to continue to perform well as it expands is important.

**Expandability** This refers to the overall ability of the solution to grow. A storage solution that expands to 50 PB is con-

sidered more expandable than a solution that only scales to 10 PB. This metric is related to, but different, from scalability, which is a measure of the solution's performance as it expands. Expandability is a major architecture factor for storage solutions with general purpose OpenStack cloud. For example, the storage architecture for a cloud that is intended for a development platform may not have the same expandability and scalability requirements as a cloud that is intended for a commercial product.

Storage hardware architecture is largely determined by the selected storage architecture. The selection of storage architecture, as well as the corresponding storage hardware, is determined by evaluating possible solutions against the critical factors, the user requirements, technical considerations, and operational considerations. A combination of all the factors and considerations will determine which approach will be best.

Using a scale-out storage solution with direct-attached storage (DAS) in the servers is well suited for a general purpose OpenStack cloud. In this scenario, it is possible to populate storage in either the compute hosts similar to a grid computing solution or into hosts dedicated to providing block storage exclusively. When deploying storage in the compute hosts, appropriate hardware which can support both the storage and compute services on the same hardware will be required. This approach is referred to as a grid computing architecture because there is a grid of modules that have both compute and storage in a single box.

Understanding the requirements of cloud services will help determine if Ceph, Gluster, or a similar scale-out solution should be used. It can then be further determined if a single, highly expandable and highly vertical, scalable, centralized storage array should be included in the design. Once the approach has been determined, the storage hardware needs to be chosen based on this criteria. If a centralized storage array fits the requirements best, then the array vendor will determine the hardware. For cost reasons it may be decided to build an open source storage array using solutions such as OpenFiler, Nexenta Open Source, or BackBlaze Open Source.

This list expands upon the potential impacts for including a particular storage architecture (and corresponding storage hardware) into the design for a general purpose OpenStack cloud:

**Connectivity** Ensure that, if storage protocols other than Ethernet are part of the storage solution, the appropriate hardware has

| | |
|---|---|
| | been selected. Some examples include InfiniBand, FDDI and Fibre Channel. If a centralized storage array is selected, ensure that the hypervisor will be able to connect to that storage array for image storage. |
| **Usage** | How the particular storage architecture will be used is critical for determining the architecture. Some of the configurations that will influence the architecture include whether it will be used by the hypervisors for ephemeral instance storage or if OpenStack Object Storage will use it for object storage. All of these usage models are affected by the selection of particular storage architecture and the corresponding storage hardware to support that architecture. |
| **Instance and image locations** | Where instances and images will be stored will influence the architecture. For example, instances can be stored in a number of options. OpenStack Block Storage is a good location for instances because it is persistent block storage, however, OpenStack Object Storage can be used if storage latency is less of a concern. The same argument applies to the appropriate image storage location. |
| **Server hardware** | If the solution is a scale-out storage architecture that includes DAS, naturally that will affect the server hardware selection. This could ripple into the decisions that affect host density, instance density, power density, OS-hypervisor, management tools and others. |

General purpose OpenStack cloud has multiple options. As a result, there is no single decision that will apply to all implementations. The key factors that will have an influence on selection of storage hardware for a general purpose OpenStack cloud are as follows:

**Capacity**

Hardware resources selected for the resource nodes should be capable of supporting enough storage for the cloud services that will use them. It is important to clearly define the initial requirements and ensure that the design can support adding capacity as resources are used in the cloud, as workloads are relatively unknown. Hardware nodes selected for object storage should be capable of supporting a large number of inexpensive disks and should not have any reliance on RAID controller cards. Hardware nodes selected for block storage should be capable of supporting higher speed storage solutions and RAID controller cards to provide performance and redundancy to storage at the hardware level. Selecting hardware RAID controllers that can automatically repair damaged arrays will further assist with replacing and repairing degraded or destroyed storage devices within the cloud.

**Performance**

Disks selected for the object storage service do not need to be fast performing disks. It is recommended that object storage nodes take advantage of the best cost per terabyte available for storage at the time of acquisition and avoid enterprise class drives. In contrast, disks chosen for the block storage service should take advantage of performance boosting features and may entail the use of SSDs or flash storage to provide for high performing block storage pools. Storage performance of ephemeral disks used for instances should also be taken into consideration. If compute pools are expected to have a high utilization of ephemeral storage or requires very high performance, it would be advantageous to deploy similar hardware solutions to block storage in order to increase the storage performance.

**Fault tolerance**

Object storage resource nodes have no requirements for hardware fault tolerance or RAID controllers. It is not necessary to plan for fault tolerance within the object storage hardware because the object storage service provides replication between zones as a feature of the service. Block storage nodes, compute nodes and cloud controllers should all have fault tolerance built in at the hard-

ware level by making use of hardware RAID controllers and varying levels of RAID configuration. The level of RAID chosen should be consistent with the performance and availability requirements of the cloud.

# Selecting networking hardware

As is the case with storage architecture, selecting a network architecture often determines which network hardware will be used. The networking software in use is determined by the selected networking hardware. Some design impacts are obvious, for example, selecting networking hardware that only supports Gigabit Ethernet (GbE) will naturally have an impact on many different areas of the overall design. Similarly, deciding to use 10 Gigabit Ethernet (10 GbE) has a number of impacts on various areas of the overall design.

As an example, selecting Cisco networking hardware implies that the architecture will be using Cisco networking software like IOS or NX-OS. Conversely, selecting Arista networking hardware means the network devices will use the Arista networking software called Extensible Operating System (EOS). In addition, there are more subtle design impacts that need to be considered. The selection of certain networking hardware (and therefore the networking software) could affect the management tools that can be used. There are exceptions to this; the rise of "open" networking software that supports a range of networking hardware means that there are instances where the relationship between networking hardware and networking software are not as tightly defined. An example of this type of software is Cumulus Linux, which is capable of running on a number of switch vendor's hardware solutions.

Some of the key considerations that should be included in the selection of networking hardware include:

**Port count** The design will require networking hardware that has the requisite port count.

**Port density** The network design will be affected by the physical space that is required to provide the requisite port count. A switch that can provide 48 10 GbE ports in 1U has a much higher port density than a switch that provides 24 10 GbE ports in 2U. A higher port density is preferred, as it leaves more rack space for compute

or storage components that may be required by the design. This can also lead into concerns about fault domains and power density that should be considered. Higher density switches are more expensive and should also be considered, as it is important not to over design the network if it is not required.

**Port speed** The networking hardware must support the proposed network speed, for example: 1 GbE, 10 GbE, or 40 GbE (or even 100 GbE).

**Redundancy** The level of network hardware redundancy required is influenced by the user requirements for high availability and cost considerations. Network redundancy can be achieved by adding redundant power supplies or paired switches. If this is a requirement, the hardware will need to support this configuration. User requirements will determine if a completely redundant network infrastructure is required.

**Power requirements** Make sure that the physical data center provides the necessary power for the selected network hardware. This is not an issue for top of rack (ToR) switches, but may be an issue for spine switches in a leaf and spine fabric, or end of row (EoR) switches.

There is no single best practice architecture for the networking hardware supporting a general purpose OpenStack cloud that will apply to all implementations. Some of the key factors that will have a strong influence on selection of networking hardware include:

**Connectivity** All nodes within an OpenStack cloud require some form of network connectivity. In some cases, nodes require access to more than one network segment. The design must encompass sufficient network capacity and bandwidth to ensure that all communications within the cloud, both north-south and east-west traffic have sufficient resources available.

**Scalability** The chosen network design should encompass a physical and logical network design that can be easily expanded upon. Network hardware should offer the ap-

propriate types of interfaces and speeds that are required by the hardware nodes.

**Availability** To ensure that access to nodes within the cloud is not interrupted, it is recommended that the network architecture identify any single points of failure and provide some level of redundancy or fault tolerance. With regard to the network infrastructure itself, this often involves use of networking protocols such as LACP, VRRP or others to achieve a highly available network connection. In addition, it is important to consider the networking implications on API availability. In order to ensure that the APIs, and potentially other services in the cloud are highly available, it is recommended to design load balancing solutions within the network architecture to accommodate for these requirements.

# Software selection

Software selection for a general purpose OpenStack architecture design needs to include these three areas:

- Operating system (OS) and hypervisor
- OpenStack components
- Supplemental software

# Operating system and hypervisor

The selection of operating system (OS) and hypervisor has a tremendous impact on the overall design. Selecting a particular operating system and hypervisor can also directly affect server hardware selection. It is recommended to make sure the storage hardware selection and topology support the selected operating system and hypervisor combination. Finally, it is important to ensure that the networking hardware selection and topology will work with the chosen operating system and hypervisor combination. For example, if the design uses Link Aggregation Control Protocol (LACP), the OS and hypervisor both need to support it.

Some areas that could be impacted by the selection of OS and hypervisor include:

**Cost** Selecting a commercially supported hypervisor, such as Microsoft Hyper-V, will result in

| | |
|---|---|
| | a different cost model rather than community-supported open source hypervisors including *KVM*, Kinstance or *Xen*. When comparing open source OS solutions, choosing Ubuntu over Red Hat (or vice versa) will have an impact on cost due to support contracts. On the other hand, business or application requirements may dictate a specific or commercially supported hypervisor. |
| **Supportability** | Depending on the selected hypervisor, the staff should have the appropriate training and knowledge to support the selected OS and hypervisor combination. If they do not, training will need to be provided which could have a cost impact on the design. |
| **Management tools** | The management tools used for Ubuntu and Kinstance differ from the management tools for VMware vSphere. Although both OS and hypervisor combinations are supported by OpenStack, there will be very different impacts to the rest of the design as a result of the selection of one combination versus the other. |
| **Scale and performance** | Ensure that selected OS and hypervisor combinations meet the appropriate scale and performance requirements. The chosen architecture will need to meet the targeted instance-host ratios with the selected OS-hypervisor combinations. |
| **Security** | Ensure that the design can accommodate the regular periodic installation of application security patches while maintaining the required workloads. The frequency of security patches for the proposed OS-hypervisor combination will have an impact on performance and the patch installation process could affect maintenance windows. |
| **Supported features** | Determine which features of OpenStack are required. This will often determine the selection of the OS-hypervisor combination. |

| | |
|---|---|
| | Certain features are only available with specific OSs or hypervisors. For example, if certain features are not available, the design might need to be modified to meet the user requirements. |
| **Interoperability** | Consideration should be given to the ability of the selected OS-hypervisor combination to interoperate or co-exist with other OS-hypervisors as well as other software solutions in the overall design (if required). Operational troubleshooting tools for one OS-hypervisor combination may differ from the tools used for another OS-hypervisor combination and, as a result, the design will need to address if the two sets of tools need to interoperate. |

# OpenStack components

The selection of which OpenStack components are included has a significant impact on the overall design. While there are certain components that will always be present, (Compute and Image Service, for example) there are other services that may not be required. As an example, a certain design might not need *Orchestration*. Omitting Orchestration would not have a significant impact on the overall design of a cloud; however, if the architecture uses a replacement for OpenStack Object Storage for its storage component, it could potentially have significant impacts on the rest of the design.

The exclusion of certain OpenStack components might also limit or constrain the functionality of other components. If the architecture includes Orchestration but excludes Telemetry, then the design will not be able to take advantage of Orchestrations' auto scaling functionality (which relies on information from Telemetry). It is important to research the component interdependencies in conjunction with the technical requirements before deciding what components need to be included and what components can be dropped from the final architecture.

# Supplemental components

While OpenStack is a fairly complete collection of software projects for building a platform for cloud services, there are invariably additional pieces of software that need to be considered in any given OpenStack design.

# Networking software

OpenStack Networking provides a wide variety of networking services for instances. There are many additional networking software packages that might be useful to manage the OpenStack components themselves. Some examples include software to provide load balancing, network redundancy protocols, and routing daemons. Some of these software packages are described in more detail in the *OpenStack High Availability Guide* (refer to the Network controller cluster stack chapter of the OpenStack High Availability Guide).

For a general purpose OpenStack cloud, the OpenStack infrastructure components will need to be highly available. If the design does not include hardware load balancing, networking software packages like HAProxy will need to be included.

# Management software

The selected supplemental software solution impacts and affects the overall OpenStack cloud design. This includes software for providing clustering, logging, monitoring and alerting.

Inclusion of clustering software, such as Corosync or Pacemaker, is determined primarily by the availability requirements. Therefore, the impact of including (or not including) these software packages is primarily determined by the availability of the cloud infrastructure and the complexity of supporting the configuration after it is deployed. The *OpenStack High Availability Guide* provides more details on the installation and configuration of Corosync and Pacemaker, should these packages need to be included in the design.

Requirements for logging, monitoring, and alerting are determined by operational considerations. Each of these sub-categories includes a number of various options. For example, in the logging sub-category one might consider Logstash, Splunk, instanceware Log Insight, or some other log aggregation-consolidation tool. Logs should be stored in a centralized location to make it easier to perform analytics against the data. Log data analytics engines can also provide automation and issue notification by providing a mechanism to both alert and automatically attempt to remediate some of the more commonly known issues.

If any of these software packages are required, then the design must account for the additional resource consumption (CPU, RAM, storage, and

network bandwidth for a log aggregation solution, for example). Some other potential design impacts include:

- OS-hypervisor combination: Ensure that the selected logging, monitoring, or alerting tools support the proposed OS-hypervisor combination.
- Network hardware: The network hardware selection needs to be supported by the logging, monitoring, and alerting software.

## Database software

A large majority of the OpenStack components require access to back-end database services to store state and configuration information. Selection of an appropriate back-end database that will satisfy the availability and fault tolerance requirements of the OpenStack services is required. OpenStack services supports connecting to any database that is supported by the SQLAlchemy python drivers, however, most common database deployments make use of MySQL or variations of it. It is recommended that the database which provides back-end service within a general purpose cloud be made highly available when using an available technology which can accomplish that goal. Some of the more common software solutions used include Galera, MariaDB and MySQL with multi-master replication.

## Addressing performance-sensitive workloads

Although one of the key defining factors for a general purpose OpenStack cloud is that performance is not a determining factor, there may still be some performance-sensitive workloads deployed on the general purpose OpenStack cloud. For design guidance on performance-sensitive workloads, it is recommended to refer to the focused scenarios later in this guide. The resource-focused guides can be used as a supplement to this guide to help with decisions regarding performance-sensitive workloads.

## Compute-focused workloads

In an OpenStack cloud that is compute-focused, there are some design choices that can help accommodate those workloads. Compute-focused workloads are generally those that would place a higher demand on CPU and memory resources with lower priority given to storage and network performance, other than what is required to support the intended compute workloads. For guidance on designing for this type of cloud, please refer to Chapter 3, “Compute focused” [51].

## Network-focused workloads

In a network-focused OpenStack cloud some design choices can improve the performance of these types of workloads. Network-focused workloads have extreme demands on network bandwidth and services that require specialized consideration and planning. For guidance on designing for this type of cloud, please refer to Chapter 5, "Network focused" [109].

## Storage-focused workloads

Storage focused OpenStack clouds need to be designed to accommodate workloads that have extreme demands on either object or block storage services that require specialized consideration and planning. For guidance on designing for this type of cloud, please refer to Chapter 4, "Storage focused" [81].

# Prescriptive example

An online classified advertising company wants to run web applications consisting of Tomcat, Nginx and MariaDB in a private cloud. In order to meet policy requirements, the cloud infrastructure will run in their own data center. They have predictable load requirements but require an element of scaling to cope with nightly increases in demand. Their current environment is not flexible enough to align with their goal of running an open source API driven environment. Their current environment consists of the following:

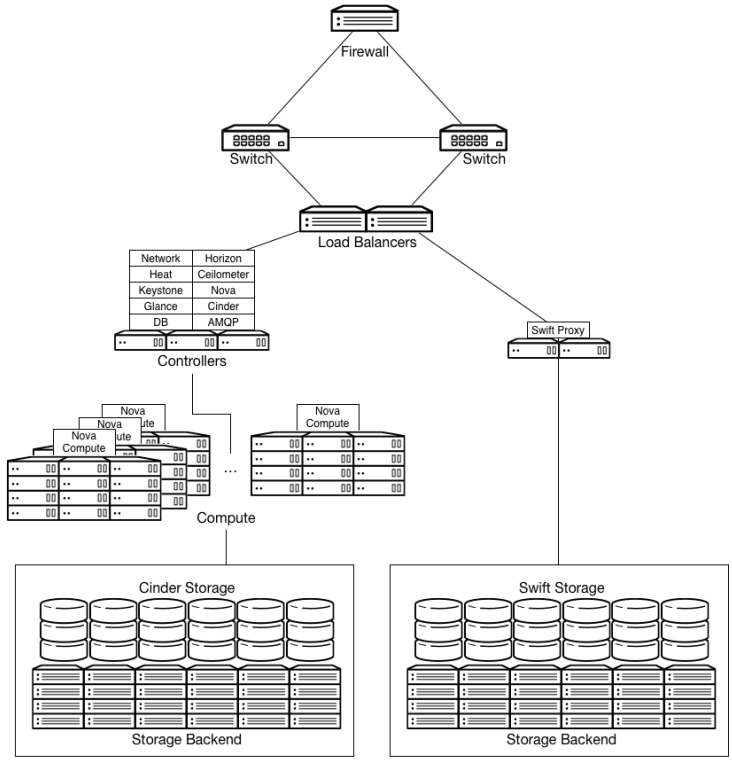
- Between 120 and 140 installations of Nginx and Tomcat, each with 2 vCPUs and 4 GB of RAM
- A three-node MariaDB and Galera cluster, each with 4 vCPUs and 8 GB RAM

The company runs hardware load balancers and multiple web applications serving the sites. The company orchestrates their environment using a combination of scripts and Puppet. The websites generate a large amount of log data each day that needs to be archived.

The solution would consist of the following OpenStack components:

- A firewall, switches and load balancers on the public facing network connections.

- OpenStack Controller services running Image, Identity, Networking and supporting services such as MariaDB and RabbitMQ. The controllers will run in a highly available configuration on at least three controller nodes.
- OpenStack Compute nodes running the KVM hypervisor.
- OpenStack Block Storage for use by compute instances that require persistent storage such as databases for dynamic sites.
- OpenStack Object Storage for serving static objects such as images.




Running up to 140 web instances and the small number of MariaDB instances requires 292 vCPUs available, as well as 584 GB RAM. On a typical 1U server using dual-socket hex-core Intel CPUs with Hyperthreading, and assuming 2:1 CPU overcommit ratio, this would require 8 OpenStack Compute nodes.

The web application instances run from local storage on each of the OpenStack Compute nodes. The web application instances are stateless, mean-

ing that any of the instances can fail and the application will continue to function.

MariaDB server instances store their data on shared enterprise storage, such as NetApp or Solidfire devices. If a MariaDB instance fails, storage would be expected to be re-attached to another instance and rejoined to the Galera cluster.

Logs from the web application servers are shipped to OpenStack Object Storage for later processing and archiving.

In this scenario, additional capabilities can be realized by moving static web content to be served from OpenStack Object Storage containers, and backing the OpenStack Image Service with OpenStack Object Storage. Note that an increase in OpenStack Object Storage means that network bandwidth needs to be taken in to consideration. It is best to run OpenStack Object Storage with network connections offering 10 GbE or better connectivity.

There is also a potential to leverage the Orchestration and Telemetry modules to provide an auto-scaling, orchestrated web application environment. Defining the web applications in *Heat Orchestration Templates (HOT)* would negate the reliance on the scripted Puppet solution currently employed.

OpenStack Networking can be used to control hardware load balancers through the use of plug-ins and the Networking API. This would allow a user to control hardware load balance pools and instances as members in these pools, but their use in production environments must be carefully weighed against current stability.

# 3. Compute focused

## Table of Contents



A compute-focused cloud is a specialized subset of the general purpose OpenStack cloud architecture. Unlike the general purpose OpenStack architecture, which is built to host a wide variety of workloads and applications and does not heavily tax any particular computing aspect, a compute-focused cloud is built and designed specifically to support compute intensive workloads. As such, the design must be specifically tailored to support hosting compute intensive workloads. Compute intensive workloads may be CPU intensive, RAM intensive, or both. However, they are not typically storage intensive or network intensive. Compute-focused workloads may include the following use cases:

- High performance computing (HPC)
- Big data analytics using Hadoop or other distributed data stores
- Continuous integration/continuous deployment (CI/CD)
- Platform-as-a-Service (PaaS)
- Signal processing for network function virtualization (NFV)

Based on the use case requirements, such clouds might need to provide additional services such as a virtual machine disk library, file or object storage, firewalls, load balancers, IP addresses, and network connectivity in the form of overlays or virtual local area networks (VLANs). A compute-focused OpenStack cloud will not typically use raw block storage services since the applications hosted on a compute-focused OpenStack cloud generally do not need persistent block storage.

# User requirements

Compute intensive workloads are defined by their high utilization of CPU, RAM, or both. User requirements will determine if a cloud must be built to accommodate anticipated performance demands.

**Cost**

Cost is not generally a primary concern for a compute-focused cloud, however some organizations might be concerned with cost avoidance. Repurposing existing resources to tackle compute-intensive tasks instead of needing to acquire additional resources may offer cost reduction opportunities.

**Time to market**

Compute-focused clouds can be used to deliver products more quickly, for example, speeding up a company's software development life cycle (SDLC) for building products and applications.

**Revenue opportunity**

Companies that are interested in building services or products that rely on the power of the compute resources will benefit from a compute-focused cloud. Examples include the analysis of large data sets (via Hadoop or Cassandra) or completing computational intensive tasks such as rendering, scientific computation, or simulations.

# Legal requirements

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.
- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.

- Data compliance—certain types of information needs to reside in certain locations due to regular issues—and more important cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

# Technical considerations

The following are some technical requirements that need to be incorporated into the architecture design.

**Performance**

If a primary technical concern is for the environment to deliver high performance capability, then a compute-focused design is an obvious choice because it is specifically designed to host compute-intensive workloads.

**Workload persistence**

Workloads can be either short-lived or long running. Short-lived workloads might include continuous integration and continuous deployment (CI-CD) jobs, where large numbers of compute instances are created simultaneously to perform a set of compute-intensive tasks. The results or artifacts are then copied from the instance into long-term storage before the instance is destroyed. Long-running workloads, like a Hadoop or high-performance computing (HPC) cluster, typically ingest large data sets, perform the computational work on those data sets, then push the results into long term storage. Unlike short-lived workloads, when the computational work is completed, they will remain idle until the next job is pushed to them. Long-running workloads are often larger and more complex, so the effort of building them is mitigated by keeping them active between jobs. Another example of long running workloads is legacy applications that typically are persistent over time.

| | |
|---|---|
| **Storage** | Workloads targeted for a compute-focused OpenStack cloud generally do not require any persistent block storage (although some usages of Hadoop with HDFS may dictate the use of persistent block storage). A shared filesystem or object store will maintain the initial data set(s) and serve as the destination for saving the computational results. By avoiding the input-output (IO) overhead, workload performance is significantly enhanced. Depending on the size of the data set(s), it might be necessary to scale the object store or shared file system to match the storage demand. |
| **User interface** | Like any other cloud architecture, a compute-focused OpenStack cloud requires an on-demand and self-service user interface. End users must be able to provision computing power, storage, networks and software simply and flexibly. This includes scaling the infrastructure up to a substantial level without disrupting host operations. |
| **Security** | Security is going to be highly dependent on the business requirements. For example, a computationally intense drug discovery application will obviously have much higher security requirements than a cloud that is designed for processing market data for a retailer. As a general start, the security recommendations and guidelines provided in the OpenStack Security Guide are applicable. |

## Operational considerations

The compute intensive cloud from the operational perspective is similar to the requirements for the general-purpose cloud. More details on operational requirements can be found in the general-purpose design section.

# Technical considerations

In a compute-focused OpenStack cloud, the type of instance workloads being provisioned heavily influences technical decision making. For example,

specific use cases that demand multiple short running jobs present different requirements than those that specify long-running jobs, even though both situations are considered "compute focused."

Public and private clouds require deterministic capacity planning to support elastic growth in order to meet user SLA expectations. Deterministic capacity planning is the path to predicting the effort and expense of making a given process consistently performant. This process is important because, when a service becomes a critical part of a user's infrastructure, the user's fate becomes wedded to the SLAs of the cloud itself. In cloud computing, a service's performance will not be measured by its average speed but rather by the consistency of its speed.

There are two aspects of capacity planning to consider: planning the initial deployment footprint, and planning expansion of it to stay ahead of the demands of cloud users.

Planning the initial footprint for an OpenStack deployment is typically done based on existing infrastructure workloads and estimates based on expected uptake.

The starting point is the core count of the cloud. By applying relevant ratios, the user can gather information about:

- The number of instances expected to be available concurrently: (overcommit fraction × cores) / virtual cores per instance
- How much storage is required: flavor disk size × number of instances

These ratios can be used to determine the amount of additional infrastructure needed to support the cloud. For example, consider a situation in which you require 1600 instances, each with 2 vCPU and 50 GB of storage. Assuming the default overcommit rate of 16:1, working out the math provides an equation of:

- 1600 = (16 x (number of physical cores)) / 2
- storage required = 50 GB x 1600

On the surface, the equations reveal the need for 200 physical cores and 80 TB of storage for `/var/lib/nova/instances/`. However, it is also important to look at patterns of usage to estimate the load that the API services, database servers, and queue servers are likely to encounter.

Consider, for example, the differences between a cloud that supports a managed web-hosting platform with one running integration tests for a

development project that creates one instance per code commit. In the former, the heavy work of creating an instance happens only every few months, whereas the latter puts constant heavy load on the cloud controller. The average instance lifetime must be considered, as a larger number generally means less load on the cloud controller.

Aside from the creation and termination of instances, the impact of users must be considered when accessing the service, particularly on nova-api and its associated database. Listing instances garners a great deal of information and, given the frequency with which users run this operation, a cloud with a large number of users can increase the load significantly. This can even occur unintentionally. For example, the OpenStack Dashboard instances tab refreshes the list of instances every 30 seconds, so leaving it open in a browser window can cause unexpected load.

Consideration of these factors can help determine how many cloud controller cores are required. A server with 8 CPU cores and 8 GB of RAM server would be sufficient for up to a rack of compute nodes, given the above caveats.

Key hardware specifications are also crucial to the performance of user instances. Be sure to consider budget and performance needs, including storage performance (spindles/core), memory availability (RAM/core), network bandwidth (Gbps/core), and overall CPU performance (CPU/core).

The cloud resource calculator is a useful tool in examining the impacts of different hardware and instance load outs. It is available at: https://github.com/noslzzp/cloud-resource-calculator/blob/master/cloud-resource-calculator.ods

# Expansion planning

A key challenge faced when planning the expansion of cloud compute services is the elastic nature of cloud infrastructure demands. Previously, new users or customers would be forced to plan for and request the infrastructure they required ahead of time, allowing time for reactive procurement processes. Cloud computing users have come to expect the agility provided by having instant access to new resources as they are required. Consequently, this means planning should be delivered for typical usage, but also more importantly, for sudden bursts in usage.

Planning for expansion can be a delicate balancing act. Planning too conservatively can lead to unexpected oversubscription of the cloud and dissatisfied users. Planning for cloud expansion too aggressively can lead to

unexpected underutilization of the cloud and funds spent on operating infrastructure that is not being used efficiently.

The key is to carefully monitor the spikes and valleys in cloud usage over time. The intent is to measure the consistency with which services can be delivered, not the average speed or capacity of the cloud. Using this information to model performance results in capacity enables users to more accurately determine the current and future capacity of the cloud.

# CPU and RAM

(Adapted from: http://docs.openstack.org/openstack-ops/content/compute_nodes.html#cpu_choice)

In current generations, CPUs have up to 12 cores. If an Intel CPU supports Hyper-Threading, those 12 cores are doubled to 24 cores. If a server is purchased that supports multiple CPUs, the number of cores is further multiplied. Hyper-Threading is Intel's proprietary simultaneous multi-threading implementation, used to improve parallelization on their CPUs. Consider enabling Hyper-Threading to improve the performance of multithreaded applications.

Whether the user should enable Hyper-Threading on a CPU depends upon the use case. For example, disabling Hyper-Threading can be beneficial in intense computing environments. Performance testing conducted by running local workloads with both Hyper-Threading on and off can help determine what is more appropriate in any particular case.

If the Libvirt/KVM hypervisor driver are the intended use cases, then the CPUs used in the compute nodes must support virtualization by way of the VT-x extensions for Intel chips and AMD-v extensions for AMD chips to provide full performance.

OpenStack enables the user to overcommit CPU and RAM on compute nodes. This allows an increase in the number of instances running on the cloud at the cost of reducing the performance of the instances. OpenStack Compute uses the following ratios by default:

- CPU allocation ratio: 16:1
- RAM allocation ratio: 1.5:1

The default CPU allocation ratio of 16:1 means that the scheduler allocates up to 16 virtual cores per physical core. For example, if a physical node has

12 cores, the scheduler sees 192 available virtual cores. With typical flavor definitions of 4 virtual cores per instance, this ratio would provide 48 instances on a physical node.

Similarly, the default RAM allocation ratio of 1.5:1 means that the scheduler allocates instances to a physical node as long as the total amount of RAM associated with the instances is less than 1.5 times the amount of RAM available on the physical node.

For example, if a physical node has 48 GB of RAM, the scheduler allocates instances to that node until the sum of the RAM associated with the instances reaches 72 GB (such as nine instances, in the case where each instance has 8 GB of RAM).

The appropriate CPU and RAM allocation ratio must be selected based on particular use cases.

# Additional hardware

Certain use cases may benefit from exposure to additional devices on the compute node. Examples might include:

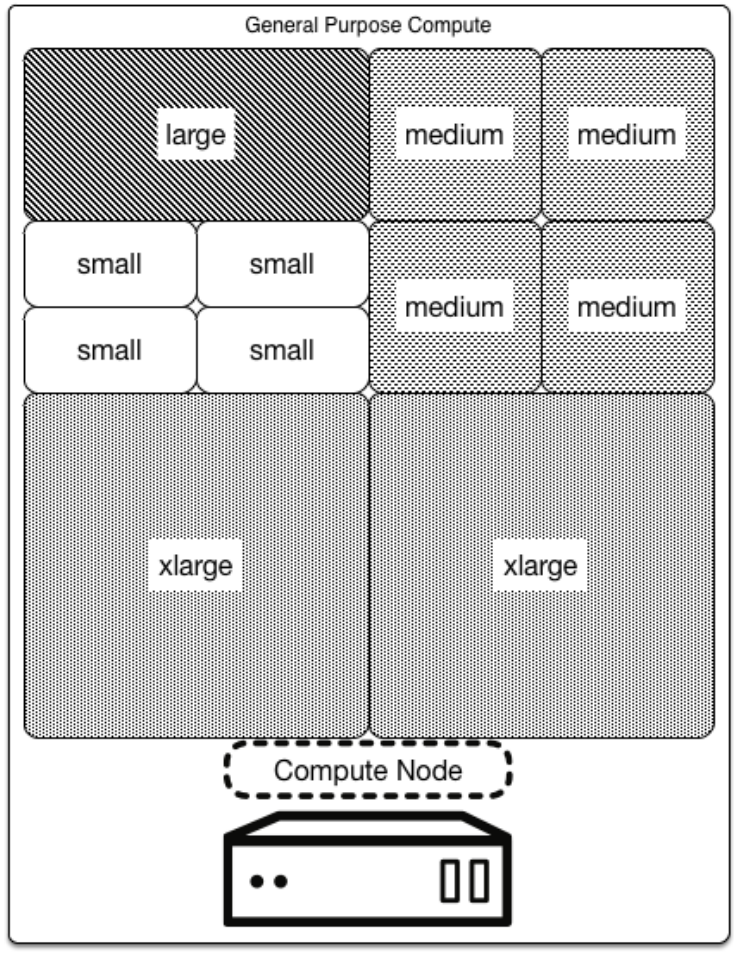
- High performance computing jobs that benefit from the availability of graphics processing units (GPUs) for general-purpose computing.
- Cryptographic routines that benefit from the availability of hardware random number generators to avoid entropy starvation.
- Database management systems that benefit from the availability of SSDs for ephemeral storage to maximize read/write time when it is required.

Host aggregates are used to group hosts that share similar characteristics, which can include hardware similarities. The addition of specialized hardware to a cloud deployment is likely to add to the cost of each node, so careful consideration must be given to whether all compute nodes, or just a subset which is targetable using flavors, need the additional customization to support the desired workloads.

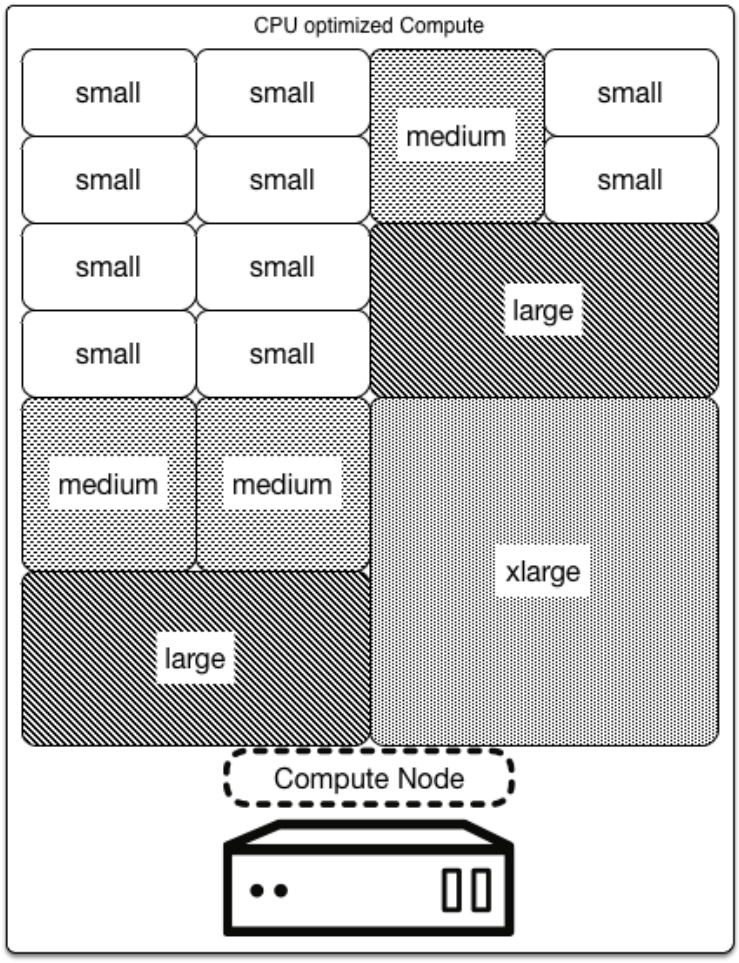
# Utilization

Infrastructure-as-a-Service offerings, including OpenStack, use flavors to provide standardized views of virtual machine resource requirements that simplify the problem of scheduling instances while making the best use of the available physical resources.

In order to facilitate packing of virtual machines onto physical hosts, the default selection of flavors are constructed so that the second largest flavor is half the size of the largest flavor in every dimension. It has half the vCPUs, half the vRAM, and half the ephemeral disk space. The next largest flavor is half that size again. As a result, packing a server for general purpose computing might look conceptually something like this figure:




On the other hand, a CPU optimized packed server might look like the following figure:




These default flavors are well suited to typical load outs for commodity server hardware. To maximize utilization, however, it may be necessary to customize the flavors or create new ones, to better align instance sizes to the available hardware.

Workload characteristics may also influence hardware choices and flavor configuration, particularly where they present different ratios of CPU versus RAM versus HDD requirements.

For more information on Flavors refer to: http://docs.openstack.org/openstack-ops/content/flavors.html

# Performance

The infrastructure of a cloud should not be shared, so that it is possible for the workloads to consume as many resources as are made available, and accommodations should be made to provide large scale workloads.

The duration of batch processing differs depending on individual workloads that are launched. Time limits range from seconds, minutes to hours, and as a result it is considered difficult to predict when resources will be used, for how long, and even which resources will be used.

# Security

The security considerations needed for this scenario are similar to those of the other scenarios discussed in this book.

A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system. Typically they have the same authentication and authorization requirements and users.

These security domains are:

1. Public
2. Guest
3. Management
4. Data

These security domains can be mapped individually to the installation, or they can also be combined. For example, some deployment topologies combine both guest and data domains onto one physical network, whereas in other cases these networks are physically separated. In each case, the cloud operator should be aware of the appropriate security concerns. Security domains should be mapped out against specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which the user has no authority. This domain should always be considered untrusted.

Typically used for compute instance-to-instance traffic, the guest security domain handles compute data generated by instances on the cloud; not services that support the operation of the cloud, for example API calls. Public cloud providers and private cloud providers who do not have stringent controls on instance use or who allow unrestricted Internet access to instances should consider this domain to be untrusted. Private cloud providers may want to consider this network as internal and therefore trusted only if they have controls in place to assert that they trust instances and all their tenants.

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, user names, and passwords. In most deployments this domain is considered trusted.

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. Much of the data that crosses this network has high integrity and confidentiality requirements and depending on the type of deployment there may also be strong availability requirements. The trust level of this network is heavily dependent on deployment decisions and as such we do not assign this any default level of trust.

When deploying OpenStack in an enterprise as a private cloud it is assumed to be behind a firewall and within the trusted network alongside existing systems. Users of the cloud are typically employees or trusted individuals that are bound by the security requirements set forth by the company. This tends to push most of the security domains towards a more trusted model. However, when deploying OpenStack in a public-facing role, no assumptions can be made and the attack vectors significantly increase. For example, the API endpoints and the software behind it will be vulnerable to potentially hostile entities wanting to gain unauthorized access or prevent access to services. This can result in loss of reputation and must be protected against through auditing and appropriate filtering.

Consideration must be taken when managing the users of the system, whether it is the operation of public or private clouds. The identity service allows for LDAP to be part of the authentication process, and includes such systems as an OpenStack deployment that may ease user management if integrated into existing systems.

It is strongly recommended that the API services are placed behind hardware that performs SSL termination. API services transmit user names, passwords, and generated tokens between client machines and API endpoints and therefore must be secured.

More information on OpenStack Security can be found at http://docs.openstack.org/security-guide/

# OpenStack components

Due to the nature of the workloads that will be used in this scenario, a number of components will be highly beneficial in a Compute-focused cloud. This includes the typical OpenStack components:

- OpenStack Compute (nova)
- OpenStack Image Service (glance)
- OpenStack Identity (keystone)

Also consider several specialized components:

- *Orchestration* module (*heat*)

It is safe to assume that, given the nature of the applications involved in this scenario, these will be heavily automated deployments. Making use of Orchestration will be highly beneficial in this case. Deploying a batch of instances and running an automated set of tests can be scripted, however it makes sense to use the Orchestration module to handle all these actions.

- Telemetry module (ceilometer)

Telemetry and the alarms it generates are required to support autoscaling of instances using Orchestration. Users that are not using the Orchestration module do not need to deploy the Telemetry module and may choose to use other external solutions to fulfill their metering and monitoring requirements.

See also: http://docs.openstack.org/openstack-ops/content/logging_monitoring.html

- OpenStack Block Storage (cinder)

Due to the burst-able nature of the workloads and the applications and instances that will be used for batch processing, this cloud will utilize mainly memory or CPU, so the need for add-on storage to each instance is not a likely requirement. This does not mean that OpenStack Block Storage (cinder) will not be used in the infrastructure, but typically it will not be used as a central component.

- Networking

When choosing a networking platform, ensure that it either works with all desired hypervisor and container technologies and their OpenStack drivers, or includes an implementation of an ML2 mechanism driver. Networking platforms that provide ML2 mechanisms drivers can be mixed.

# Operational considerations

Operationally, there are a number of considerations that affect the design of compute-focused OpenStack clouds. Some examples might include enforcing strict API availability requirements, understanding and dealing with failure scenarios, or managing host maintenance schedules.

Service-level agreements (SLAs) are a contractual obligation that gives assurances around availability of a provided service. As such, factoring in promises of availability implies a certain level of redundancy and resiliency when designing an OpenStack cloud.

- Guarantees for API availability imply multiple infrastructure services combined with appropriately high available load balancers.
- Network uptime guarantees will affect the switch design and might require redundant switching and power.
- Network security policy requirements need to be factored in to deployments.

Knowing when and where to implement redundancy and high availability (HA) is directly affected by terms contained in any associated SLA, if one is present.

# Support and maintainability

OpenStack cloud management requires operations staff to be able to understand and comprehend design architecture content on some level. The level of skills and the level of separation of the operations and engineering staff is dependent on the size and purpose of the installation. A large cloud service provider or a telecom provider is more inclined to be managed by specially trained dedicated operations organization. A smaller implementation is more inclined to rely on a smaller support staff that might need to take on the combined engineering, design and operations functions.

Maintaining OpenStack installations require a variety of technical skills. Some of these skills may include the ability to debug Python log output to a basic level as well as an understanding of networking concepts.

Consider incorporating features into the architecture and design that reduce the operational burden. Some examples include automating some of the operations functions, or alternatively exploring the possibility of using a third party management company with special expertise in managing OpenStack deployments.

## Monitoring

Like any other infrastructure deployment, OpenStack clouds need an appropriate monitoring platform to ensure errors are caught and managed appropriately. Consider leveraging any existing monitoring system to see if it will be able to effectively monitor an OpenStack environment. While there are many aspects that need to be monitored, specific metrics that are critically important to capture include image disk utilization, or response time to the Compute API.

## Expected and unexpected server downtime

At some point, servers will fail. The SLAs in place affect how the design has to address recovery time. Recovery of a failed host may mean restoring instances from a snapshot, or respawning that instance on another available host, which then has consequences on the overall application design running on the OpenStack cloud.

It might be acceptable to design a compute-focused cloud without the ability to migrate instances from one host to another, because the expectation is that the application developer must handle failure within the application itself. Conversely, a compute-focused cloud might be provisioned to provide extra resilience as a requirement of that business. In this scenario, it is expected that extra supporting services are also deployed, such as shared storage attached to hosts to aid in recovery and resiliency of services in order to meet strict SLAs.

## Capacity planning

Adding extra capacity to an OpenStack cloud is an easy horizontally scaling process, as consistently configured nodes automatically attach to an OpenStack cloud. Be mindful, however, of any additional work to place the nodes into appropriate Availability Zones and Host Aggregates if nec-

essary. The same (or very similar) CPUs are recommended when adding extra nodes to the environment because it reduces the chance to break any live-migration features if they are present. Scaling out hypervisor hosts also has a direct effect on network and other data center resources, so factor in this increase when reaching rack capacity or when extra network switches are required.

Compute hosts can also have internal components changed to account for increases in demand, a process also known as vertical scaling. Swapping a CPU for one with more cores, or increasing the memory in a server, can help add extra needed capacity depending on whether the running applications are more CPU intensive or memory based (as would be expected in a compute-focused OpenStack cloud).

Another option is to assess the average workloads and increase the number of instances that can run within the compute environment by adjusting the overcommit ratio. While only appropriate in some environments, it's important to remember that changing the CPU overcommit ratio can have a detrimental effect and cause a potential increase in noisy neighbor. The added risk of increasing the overcommit ratio is more instances will fail when a compute host fails. In a compute-focused OpenStack design architecture, increasing the CPU overcommit ratio increases the potential for noisy neighbor issues and is not recommended.

# Architecture

The hardware selection covers three areas:

- Compute
- Network
- Storage

In a compute-focused OpenStack cloud the hardware selection must reflect the workloads being compute intensive. Compute-focused is defined as having extreme demands on processor and memory resources. The hardware selection for a compute-focused OpenStack architecture design must reflect this preference for compute-intensive workloads, as these workloads are not storage intensive, nor are they consistently network intensive. The network and storage may be heavily utilized while loading a data set into the computational cluster, but they are not otherwise intensive.

Compute (server) hardware must be evaluated against four opposing dimensions:

| | |
|---|---|
| **Server density** | A measure of how many servers can fit into a given measure of physical space, such as a rack unit [U]. |
| **Resource capacity** | The number of CPU cores, how much RAM, or how much storage a given server will deliver. |
| **Expandability** | The number of additional resources that can be added to a server before it has reached its limit. |
| **Cost** | The relative purchase price of the hardware weighted against the level of design effort needed to build the system. |

The dimensions need to be weighted against each other to determine the best design for the desired purpose. For example, increasing server density means sacrificing resource capacity or expandability. Increasing resource capacity and expandability can increase cost but decreases server density. Decreasing cost can mean decreasing supportability, server density, resource capacity, and expandability.

Selection of hardware for a compute-focused cloud should have an emphasis on server hardware that can offer more CPU sockets, more CPU cores, and more RAM; network connectivity and storage capacity are less critical. The hardware will need to be configured to provide enough network connectivity and storage capacity to meet minimum user requirements, but they are not the primary consideration.

Some server hardware form factors are better suited than others, as CPU and RAM capacity have the highest priority.

- Most blade servers can support dual-socket multi-core CPUs. To avoid the limit means selecting "full width" or "full height" blades, which consequently loses server density. As an example, using high density blade servers including HP BladeSystem and Dell PowerEdge M1000e) which support up to 16 servers in only 10 rack units using half-height blades, suddenly decreases the density by 50% by selecting full-height blades resulting in only 8 servers per 10 rack units.

- 1U rack-mounted servers (servers that occupy only a single rack unit) may be able to offer greater server density than a blade server solution. It is possible to place 40 servers in a rack, providing space for the top of

rack [ToR] switches, versus 32 "full width" or "full height" blade servers in a rack), but often are limited to dual-socket, multi-core CPU configurations. Note that, as of the Icehouse release, neither HP, IBM, nor Dell offered 1U rack servers with more than 2 CPU sockets. To obtain greater than dual-socket support in a 1U rack-mount form factor, customers need to buy their systems from Original Design Manufacturers (ODMs) or second-tier manufacturers. This may cause issues for organizations that have preferred vendor policies or concerns with support and hardware warranties of non-tier 1 vendors.

- 2U rack-mounted servers provide quad-socket, multi-core CPU support, but with a corresponding decrease in server density (half the density offered by 1U rack-mounted servers).

- Larger rack-mounted servers, such as 4U servers, often provide even greater CPU capacity, commonly supporting four or even eight CPU sockets. These servers have greater expandability, but such servers have much lower server density and usually greater hardware cost.

- "Sled servers" (rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure) deliver increased density as compared to typical 1U or 2U rack-mounted servers. For example, many sled servers offer four independent dual-socket nodes in 2U for a total of 8 CPU sockets in 2U. However, the dual-socket limitation on individual nodes may not be sufficient to offset their additional cost and configuration complexity.

The following facts will strongly influence server hardware selection for a compute-focused OpenStack design architecture:

**Instance density** — In this architecture instance density is considered lower; therefore CPU and RAM over-subscription ratios are also lower. More hosts will be required to support the anticipated scale due to instance density being lower, especially if the design uses dual-socket hardware designs.

**Host density** — Another option to address the higher host count that might be needed with dual socket designs is to use a quad socket platform. Taking this approach will decrease host density, which increases rack count. This configuration

may affect the network requirements, the number of power connections, and possibly impact the cooling requirements.

**Power and cooling density** The power and cooling density requirements might be lower than with blade, sled, or 1U server designs because of lower host density (by using 2U, 3U or even 4U server designs). For data centers with older infrastructure, this may be a desirable feature.

Compute-focused OpenStack design architecture server hardware selection results in a "scale up" versus "scale out" decision. Selecting a better solution, smaller number of larger hosts, or a larger number of smaller hosts depends on a combination of factors: cost, power, cooling, physical rack and floor space, support-warranty, and manageability.

# Storage hardware selection

For a compute-focused OpenStack design architecture, the selection of storage hardware is not critical as it is not primary criteria, however it is still important. There are a number of different factors that a cloud architect must consider:

**Cost** The overall cost of the solution will play a major role in what storage architecture (and resulting storage hardware) is selected.

**Performance** The performance of the solution is also a big role and can be measured by observing the latency of storage I-O requests. In a compute-focused OpenStack cloud, storage latency can be a major consideration. In some compute-intensive workloads, minimizing the delays that the CPU experiences while fetching data from the storage can have a significant impact on the overall performance of the application.

**Scalability** This section will refer to the term "scalability" to refer to how well the storage solution performs as it is expanded up to its maximum size. A storage solution that performs well in small configurations but has degrading performance as it expands would not be con-

sidered scalable. On the other hand, a solution that continues to perform well at maximum expansion would be considered scalable.

**Expandability** Expandability refers to the overall ability of the solution to grow. A storage solution that expands to 50 PB is considered more expandable than a solution that only scales to 10PB. Note that this metric is related to, but different from, scalability, which is a measure of the solution's performance as it expands.

For a compute-focused OpenStack cloud, latency of storage is a major consideration. Using solid-state disks (SSDs) to minimize latency for instance storage and reduce CPU delays caused by waiting for the storage will increase performance. Consider using RAID controller cards in compute hosts to improve the performance of the underlying disk subsystem.

The selection of storage architecture, and the corresponding storage hardware (if there is the option), is determined by evaluating possible solutions against the key factors listed above. This will determine if a scale-out solution (such as Ceph, GlusterFS, or similar) should be used, or if a single, highly expandable and scalable centralized storage array would be a better choice. If a centralized storage array is the right fit for the requirements, the hardware will be determined by the array vendor. It is also possible to build a storage array using commodity hardware with Open Source software, but there needs to be access to people with expertise to build such a system. Conversely, a scale-out storage solution that uses direct-attached storage (DAS) in the servers may be an appropriate choice. If so, then the server hardware needs to be configured to support the storage solution.

The following lists some of the potential impacts that may affect a particular storage architecture, and the corresponding storage hardware, of a compute-focused OpenStack cloud:

**Connectivity** Based on the storage solution selected, ensure the connectivity matches the storage solution requirements. If a centralized storage array is selected, it is important to determine how the hypervisors will connect to the storage array. Connectivity could affect latency and thus performance, so check that the network characteristics will minimize latency to boost the overall performance of the design.

**Latency** Determine if the use case will have consistent or highly variable latency.

**Throughput** To improve overall performance, make sure that the storage solution throughout is optimized. While it is not likely that a compute-focused cloud will have major data I-O to and from storage, this is an important factor to consider.

**Server Hardware** If the solution uses DAS, this impacts, and is not limited to, the server hardware choice that will ripple into host density, instance density, power density, OS-hypervisor, and management tools.

Where instances need to be made highly available, or they need to be capable of migration between hosts, use of a shared storage file-system to store instance ephemeral data should be employed to ensure that compute services can run uninterrupted in the event of a node failure.

# Selecting networking hardware

Some of the key considerations that should be included in the selection of networking hardware include:

**Port count** The design will require networking hardware that has the requisite port count.

**Port density** The network design will be affected by the physical space that is required to provide the requisite port count. A switch that can provide 48 10 GbE ports in 1U has a much higher port density than a switch that provides 24 10 GbE ports in 2U. A higher port density is preferred, as it leaves more rack space for compute or storage components that might be required by the design. This also leads into concerns about fault domains and power density that must also be considered. Higher density switches are more expensive and should also be considered, as it is important not to over design the network if it is not required.

**Port speed** The networking hardware must support the proposed network speed, for example: 1 GbE, 10 GbE, or 40 GbE (or even 100 GbE).

**Redundancy** The level of network hardware redundancy required is influenced by the user require-

| | |
|---|---|
| | ments for high availability and cost considerations. Network redundancy can be achieved by adding redundant power supplies or paired switches. If this is a requirement, the hardware will need to support this configuration. User requirements will determine if a completely redundant network infrastructure is required. |
| **Power requirements** | Ensure that the physical data center provides the necessary power for the selected network hardware. This is not an issue for top of rack (ToR) switches, but may be an issue for spine switches in a leaf and spine fabric, or end of row (EoR) switches. |

It is important to first understand additional factors as well as the use case because these additional factors heavily influence the cloud network architecture. Once these key considerations have been decided, the proper network can be designed to best serve the workloads being placed in the cloud.

It is recommended that the network architecture is designed using a scalable network model that makes it easy to add capacity and bandwidth. A good example of such a model is the leaf-spline model. In this type of network design, it is possible to easily add additional bandwidth as well as scale out to additional racks of gear. It is important to select network hardware that will support the required port count, port speed and port density while also allowing for future growth as workload demands increase. It is also important to evaluate where in the network architecture it is valuable to provide redundancy. Increased network availability and redundancy comes at a cost, therefore it is recommended to weigh the cost versus the benefit gained from utilizing and deploying redundant network switches and using bonded interfaces at the host level.

# Software selection

Selecting software to be included in a compute-focused OpenStack architecture design must include three main areas:

- Operating system (OS) and hypervisor
- OpenStack components
- Supplemental software

Design decisions made in each of these areas impact the rest of the OpenStack architecture design.

# Operating system and hypervisor

The selection of operating system (OS) and hypervisor has a significant impact on the end point design. Selecting a particular operating system and hypervisor could affect server hardware selection. For example, a selected combination needs to be supported on the selected hardware. Ensuring the storage hardware selection and topology supports the selected operating system and hypervisor combination should also be considered. Additionally, make sure that the networking hardware selection and topology will work with the chosen operating system and hypervisor combination. For example, if the design uses Link Aggregation Control Protocol (LACP), the hypervisor needs to support it.

Some areas that could be impacted by the selection of OS and hypervisor include:

**Cost**

Selecting a commercially supported hypervisor such as Microsoft Hyper-V will result in a different cost model rather than chosen a community-supported open source hypervisor like Kinstance or Xen. Even within the ranks of open source solutions, choosing Ubuntu over Red Hat (or vice versa) will have an impact on cost due to support contracts. On the other hand, business or application requirements might dictate a specific or commercially supported hypervisor.

**Supportability**

Depending on the selected hypervisor, the staff should have the appropriate training and knowledge to support the selected OS and hypervisor combination. If they do not, training will need to be provided which could have a cost impact on the design.

**Management tools**

The management tools used for Ubuntu and Kinstance differ from the management tools for VMware vSphere. Although both OS and hypervisor combinations are supported by OpenStack, there will be very different impacts to the rest of the design as a

| | |
|---|---|
| | result of the selection of one combination versus the other. |
| **Scale and performance** | Ensure that selected OS and hypervisor combinations meet the appropriate scale and performance requirements. The chosen architecture will need to meet the targeted instance-host ratios with the selected OS-hypervisor combination. |
| **Security** | Ensure that the design can accommodate the regular periodic installation of application security patches while maintaining the required workloads. The frequency of security patches for the proposed OS-hypervisor combination will have an impact on performance and the patch installation process could affect maintenance windows. |
| **Supported features** | Determine what features of OpenStack are required. This will often determine the selection of the OS-hypervisor combination. Certain features are only available with specific OSs or hypervisors. For example, if certain features are not available, the design might need to be modified to meet the user requirements. |
| **Interoperability** | Consideration should be given to the ability of the selected OS-hypervisor combination to interoperate or co-exist with other OS-hypervisors, or other software solutions in the overall design (if required). Operational and troubleshooting tools for one OS-hypervisor combination may differ from the tools used for another OS-hypervisor combination and, as a result, the design will need to address if the two sets of tools need to interoperate. |

# OpenStack components

The selection of which OpenStack components will actually be included in the design and deployed has significant impact. There are certain compo-

nents that will always be present, (Compute and Image Service, for example) yet there are other services that might not need to be present. For example, a certain design may not require the Orchestration module. Omitting Heat would not typically have a significant impact on the overall design. However, if the architecture uses a replacement for OpenStack Object Storage for its storage component, this could potentially have significant impacts on the rest of the design.

For a compute-focused OpenStack design architecture, the following components would be used:

- Identity (keystone)
- Dashboard (horizon)
- Compute (nova)
- Object Storage (swift, ceph or a commercial solution)
- Image (glance)
- Networking (neutron)
- Orchestration (heat)

OpenStack Block Storage would potentially not be incorporated into a compute-focused design due to persistent block storage not being a significant requirement for the types of workloads that would be deployed onto instances running in a compute-focused cloud. However, there may be some situations where the need for performance dictates that a block storage component be used to improve data I-O.

The exclusion of certain OpenStack components might also limit or constrain the functionality of other components. If a design opts to include the Orchestration module but exclude the Telemetry module, then the design will not be able to take advantage of Orchestration's auto scaling functionality (which relies on information from Telemetry). Due to the fact that you can use Orchestration to spin up a large number of instances to perform the compute-intensive processing, including Orchestration in a compute-focused architecture design is strongly recommended.

# Supplemental software

While OpenStack is a fairly complete collection of software projects for building a platform for cloud services, there are invariably additional pieces of software that might need to be added to any given OpenStack design.

## Networking software

OpenStack Networking provides a wide variety of networking services for instances. There are many additional networking software packages that might be useful to manage the OpenStack components themselves. Some examples include software to provide load balancing, network redundancy protocols, and routing daemons. Some of these software packages are described in more detail in the *OpenStack High Availability Guide* (http://docs.openstack.org/high-availability-guide/content).

For a compute-focused OpenStack cloud, the OpenStack infrastructure components will need to be highly available. If the design does not include hardware load balancing, networking software packages like HAProxy will need to be included.

## Management software

The selected supplemental software solution impacts and affects the overall OpenStack cloud design. This includes software for providing clustering, logging, monitoring and alerting.

Inclusion of clustering Software, such as Corosync or Pacemaker, is determined primarily by the availability design requirements. Therefore, the impact of including (or not including) these software packages is primarily determined by the availability of the cloud infrastructure and the complexity of supporting the configuration after it is deployed. The OpenStack High Availability Guide provides more details on the installation and configuration of Corosync and Pacemaker, should these packages need to be included in the design.

Requirements for logging, monitoring, and alerting are determined by operational considerations. Each of these sub-categories includes a number of various options. For example, in the logging sub-category one might consider Logstash, Splunk, Log Insight, or some other log aggregation-consolidation tool. Logs should be stored in a centralized location to make it easier to perform analytics against the data. Log data analytics engines can also provide automation and issue notification by providing a mechanism to both alert and automatically attempt to remediate some of the more commonly known issues.

If any of these software packages are needed, then the design must account for the additional resource consumption (CPU, RAM, storage, and network bandwidth for a log aggregation solution, for example). Some other potential design impacts include:

- OS-hypervisor combination: Ensure that the selected logging, monitoring, or alerting tools support the proposed OS-hypervisor combination.
- Network hardware: The network hardware selection needs to be supported by the logging, monitoring, and alerting software.

### Database software

A large majority of the OpenStack components require access to back-end database services to store state and configuration information. Selection of an appropriate back-end database that will satisfy the availability and fault tolerance requirements of the OpenStack services is required. OpenStack services support connecting to any database that is supported by the SQLAlchemy Python drivers, however most common database deployments make use of MySQL or some variation of it. It is recommended that the database which provides back-end service within a general purpose cloud be made highly available using an available technology which can accomplish that goal. Some of the more common software solutions used include Galera, MariaDB and MySQL with multi-master replication.

## Prescriptive examples

The Conseil Européen pour la Recherche Nucléaire (CERN), also known as the European Organization for, Nuclear Research provides particle accelerators and other infrastructure for high-energy physics research.

As of 2011 CERN operated these two compute centers in Europe with plans to add a third.

| Data center | Approximate capacity |
|---|---|
| Geneva, Switzerland | • 3.5 Mega Watts<br>• 91000 cores<br>• 120 PB HDD<br>• 100 PB Tape<br>• 310 TB Memory |
| Budapest, Hungary | • 2.5 Mega Watts<br>• 20000 cores<br>• 6 PB HDD |

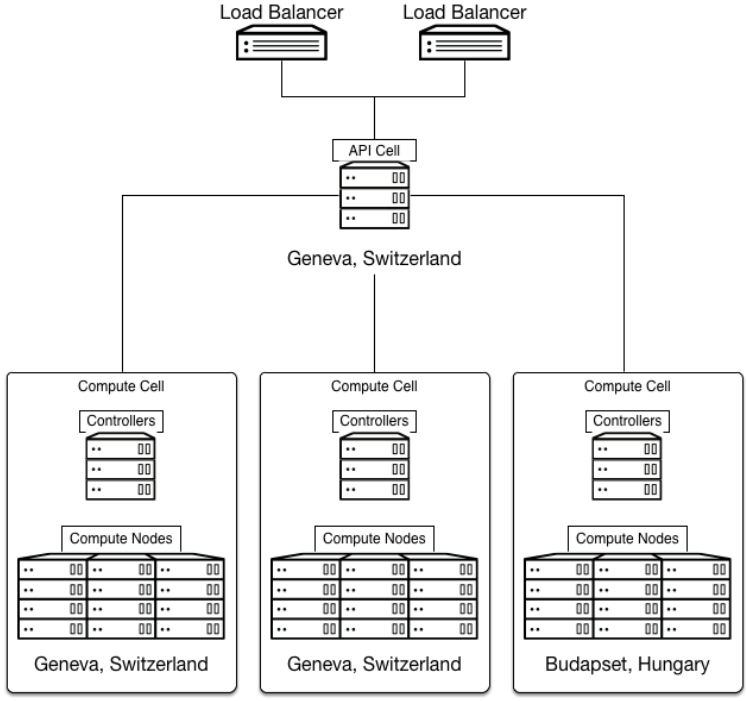
To support a growing number of compute heavy users of experiments related to the Large Hadron Collider (LHC) CERN ultimately elected to deploy an OpenStack cloud using Scientific Linux and RDO. This effort aimed to

simplify the management of the center's compute resources with a view to doubling compute capacity through the addition of an additional data center in 2013 while maintaining the same levels of compute staff.

The CERN solution uses *cells* for segregation of compute resources and to transparently scale between different data centers. This decision meant trading off support for security groups and live migration. In addition some details like flavors needed to be manually replicated across cells. In spite of these drawbacks cells were determined to provide the required scale while exposing a single public API endpoint to users.

A compute cell was created for each of the two original data centers and a third was created when a new data center was added in 2013. Each cell contains three availability zones to further segregate compute resources and at least three RabbitMQ message brokers configured to be clustered with mirrored queues for high availability.

The API cell, which resides behind a HAProxy load balancer, is located in the data center in Switzerland and directs API calls to compute cells using a customized variation of the cell scheduler. The customizations allow certain workloads to be directed to a specific data center or "all" data centers with cell selection determined by cell RAM availability in the latter case.



There is also some customization of the filter scheduler that handles placement within the cells:

- ImagePropertiesFilter - To provide special handling depending on the guest operating system in use (Linux-based or Windows-based).
- ProjectsToAggregateFilter - To provide special handling depending on the project the instance is associated with.
- default_schedule_zones - Allows the selection of multiple default availability zones, rather than a single default.

The MySQL database server in each cell is managed by a central database team and configured in an active/passive configuration with a NetApp storage back end. Backups are performed ever 6 hours.

# Network architecture

To integrate with existing CERN networking infrastructure customizations were made to legacy networking (nova-network). This was in the form of a driver to integrate with CERN's existing database for tracking MAC and IP address assignments.

The driver facilitates selection of a MAC address and IP for new instances based on the compute node the scheduler places the instance on

The driver considers the compute node that the scheduler placed an instance on and then selects a MAC address and IP from the pre-registered list associated with that node in the database. The database is then updated to reflect the instance the addresses were assigned to.

# Storage architecture

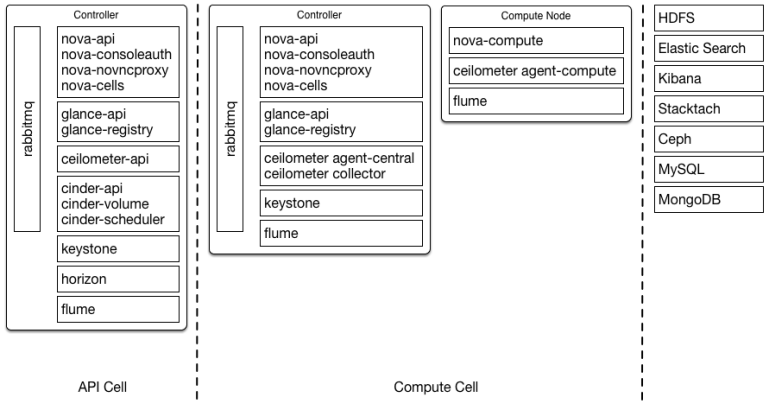
The OpenStack Image Service is deployed in the API cell and configured to expose version 1 (V1) of the API. As a result the image registry is also required. The storage back end in use is a 3 PB Ceph cluster.

A small set of "golden" Scientific Linux 5 and 6 images are maintained which applications can in turn be placed on using orchestration tools. Puppet is used for instance configuration management and customization but Orchestration deployment is expected.

# Monitoring

Although direct billing is not required, the Telemetry module is used to perform metering for the purposes of adjusting project quotas. A shard-

ed, replicated, MongoDB back end is used. To spread API load, instances of the nova-api service were deployed within the child cells for Telemetry to query against. This also meant that some supporting services including keystone, glance-api and glance-registry needed to also be configured in the child cells.




Additional monitoring tools in use include Flume, Elastic Search, Kibana, and the CERN developed Lemon project.

# References

The authors of the Architecture Design Guide would like to thank CERN for publicly documenting their OpenStack deployment in these resources, which formed the basis for this chapter:

- http://openstack-in-production.blogspot.fr
- Deep dive into the CERN Cloud Infrastructure

# 4. Storage focused

## Table of Contents



Cloud storage is a model of data storage where digital data is stored in logical pools and physical storage that spans across multiple servers and locations. Cloud storage commonly refers to a hosted object storage service, however the term has extended to include other types of data storage that are available as a service, for example block storage.

Cloud storage is based on virtualized infrastructure and resembles broader cloud computing in terms of accessible interfaces, elasticity, scalability, multi-tenancy, and metered resources. Cloud storage services can be utilized from an off-premises service or deployed on-premises.

Cloud storage is made up of many distributed, yet still synonymous resources, and is often referred to as integrated storage clouds. Cloud storage is highly fault tolerant through redundancy and the distribution of data. It is highly durable through the creation of versioned copies, and can be consistent with regard to data replicas.

At a certain scale, management of data operations can become a resource intensive process to an organization. Hierarchical storage management (HSM) systems and data grids can help annotate and report a baseline data valuation to make intelligent decisions and automate data decisions. HSM allows for automating tiering and movement, as well as orchestration of data operations. A data grid is an architecture, or set of services evolving technology, that brings together sets of services allowing users to manage large data sets.

Examples of applications that can be deployed with cloud storage characteristics are:

- Active archive, backups and hierarchical storage management.
- General content storage and synchronization. An example of this is private dropbox.

- Data analytics with parallel file systems.
- Unstructured data store for services. For example, social media back-end storage.
- Persistent block storage.
- Operating system and application image store.
- Media streaming.
- Databases.
- Content distribution.
- Cloud storage peering.

# User requirements

Storage-focused clouds are defined by their requirements for data including, but not limited to, performance, access patterns, and data structures. A balance between cost and user requirements dictate what methods and technologies will be used in a cloud architecture.

**Cost** The user pays only for the storage they actually use. This limit typically reflects average user consumption during a month. This does not mean that cloud storage is less expensive, only that it incurs operating expenses rather the capital expenses. From a business perspective, it should be beneficial for the solution to scale properly to prevent the up-front purchase of a large amount of storage that goes underutilized.

**Legal requirements** Multiple jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include data retention policies and data ownership policies.

# Legal requirements

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

| | |
|---|---|
| **Data retention** | Policies ensuring storage of persistent data and records management to meet data archival requirements. |
| **Data ownership** | Policies governing the possession and responsibility for data. |
| **Data sovereignty** | Policies governing the storage of data in foreign countries or otherwise separate jurisdictions. |
| **Data compliance** | Policies governing types of information that are required to reside in certain locations due to regular issues and cannot reside in other locations for the same reason. |

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

# Technical requirements

The following are some technical requirements that could be incorporated into the architecture design.

| | |
|---|---|
| **Storage proximity** | In order to provide high performance or large amounts of storage space the design may have to accommodate storage that is each of the hypervisors or served from a central storage device. |
| **Performance** | To boost performance the organization may want to make use of different technologies to cache disk activity. |
| **Availability** | Specific requirements regarding availability will influence the technology used to store and protect data. These requirements will be influence the cost and solution that will be implemented. |
| **Security** | Data will need to be protected both in transit and at rest. |

# Technical considerations

Some of the key technical considerations that are critical to a storage focused OpenStack design architecture include:

| | |
|---|---|
| **Input-output requirements** | Input-Output performance requirements need to be researched and modeled before deciding on a final storage framework. Running benchmarks for Input-Output performance will help provide a baseline for expected performance levels. If these tests include details, for example, object size in an object storage system or varying capacity levels for both object storage and block storage, then the resulting data can help model behavior and results during different workloads. Running scripted smaller benchmarks during the life cycle of the architecture helps record the system health at different points in time. The data from these scripted benchmarks will assist in future scoping and gaining a deeper understanding of an organization's needs. |
| **Scale** | The scale of the storage solution in a storage focused OpenStack architecture design is driven both by initial requirements, including *IOPS*, capacity, and bandwidth, as well as future needs. Planning capacity based on projected needs over the course of a budget cycle is important for a design. Ideally, the chosen architecture should balance cost and capacity while also allowing enough flexibility for new technologies and methods to be implemented as they become available. |
| **Security** | Security design around data has multiple points of focus that vary depending on SLAs, legal requirements, industry regulations, and certifications needed for systems or people. HIPPA, ISO9000, and SOX compliance should be considered based on the type of data. Levels of access control can be important for certain organizations. |

| | |
|---|---|
| **OpenStack compatibility** | Interoperability and integration with OpenStack can be paramount in deciding on a storage hardware and storage management platform. Interoperability and integration includes factors such as OpenStack Block Storage interoperability, OpenStack Object Storage compatibility, and hypervisor compatibility (which affects the ability to use storage for ephemeral instance storage). |
| **Storage management** | A range of storage management-related considerations must be addressed in the design of a storage focused OpenStack cloud. These considerations include, but are not limited to, backup strategy (and restore strategy, since a backup that can not be restored is useless), data valuation-hierarchical storage management, retention strategy, data placement, and workflow automation. |
| **Data grids** | Data grids can be helpful in deterministically answering questions around data valuation. A fundamental challenge of today's information sciences is determining which data is worth keeping, on what tier of access and performance should it reside, and how long should it remain in a storage system. Data grids improve decision making through correlation of access patterns, ownership, and business-unit revenue with other metadata values to deliver actionable information about data. |

Strive to build a flexible design that is based on a industry standard core. One way of accomplishing this might be through the use of different back ends serving different use cases.

# Operational considerations

- Maintenance tasks: The storage solution should take into account storage maintenance and the impact on underlying workloads.
- Reliability and availability: Reliability and availability depends on wide area network availability and on the level of precautions taken by the service provider.
- Flexibility: Organizations need to have the flexibility to choose between off-premise and on-premise cloud storage options. This concept relies on relevant decision criteria that is complementary to initial direct cost savings potential. For example, continuity of operations, disaster recovery, security, and records retention laws, regulations, and policies.

In a cloud environment with very high demands on storage resources, it becomes vitally important to ensure that monitoring and alerting services have been installed and configured to provide a real-time view into the health and performance of the storage systems. Using an integrated management console, or other dashboards capable of visualizing SNMP data, will be helpful in discovering and resolving issues that might arise within the storage cluster. An example of this is Ceph's Calamari.

A storage-focused cloud design should include:

- Monitoring of physical hardware resources.
- Monitoring of environmental resources such as temperature and humidity.
- Monitoring of storage resources such as available storage, memory and CPU.
- Monitoring of advanced storage performance data to ensure that storage systems are performing as expected.
- Monitoring of network resources for service disruptions which would affect access to storage.
- Centralized log collection.
- Log analytics capabilities.
- Ticketing system (or integration with a ticketing system) to track issues.

- Alerting and notification of responsible teams or automated systems which will remediate problems with storage as they arise.
- Network Operations Center (NOC) staffed and always available to resolve issues.

# Management efficiency

When designing a storage solution at scale, ease of management of the storage devices is an important consideration. Within a storage cluster, operations personnel will often be required to replace failed drives or nodes and provide ongoing maintenance of the storage hardware. When the cloud is designed and planned properly, the process can be performed in a seamless and organized manner that does not have an impact on operational efficiency.

Provisioning and configuration of new or upgraded storage is another important consideration when it comes to management of resources. The ability to easily deploy, configure, and manage storage hardware will result in a solution which is easy to manage. This also makes use of management systems that can automate other pieces of the overall solution. For example, replication, retention, data backup and recovery.

# Application awareness

When designing applications that will leverage storage solutions in the cloud, design the application to be aware of the underlying storage subsystem and the features available. As an example, when creating an application that requires replication, it is recommended that the application can detect whether replication is a feature natively available in the storage subsystem. In the event that replication is not available, the application can be designed to react in a manner such that it will be able to provide its own replication services. An application that is designed to be aware of the underlying storage systems can be deployed in a wide variety of infrastructures and still have the same basic behavior regardless of the differences in the underlying infrastructure.

# Fault tolerance and availability

Designing for fault tolerance and availability of storage systems in an OpenStack cloud is vastly different when comparing the block storage and object storage services. The object storage service is designed to have consistency and partition tolerance as a function of the application. Therefore,

it does not have any reliance on hardware RAID controllers to provide redundancy for physical disks. In contrast, block storage resource nodes are commonly configured with advanced RAID controllers and high performance disks that are designed to provide fault tolerance at the hardware level.

In cases where applications require extreme performance out of block storage devices, it is recommended to deploy high performing storage solutions such as SSD disk drives or flash storage systems. When considering these solutions, it is important to consider the availability of software and support to ease with the hardware and software integration process.

In environments that place extreme demands on block storage, it is advisable to take advantage of multiple storage pools. In this case, each pool of devices should have a similar hardware design and disk configuration across all hardware nodes in that pool. This allows for a design that provides applications with access to a wide variety of block storage pools, each with their own redundancy, availability, and performance characteristics. When deploying multiple pools of storage it is also important to consider the impact on the block storage scheduler which is responsible for provisioning storage across resource nodes. Ensuring that applications can schedule volumes in multiple regions, each with their own network, power, and cooling infrastructure, can give tenants the ability to build fault tolerant applications that will be distributed across multiple availability zones.

In addition to the block storage resource nodes, it is important to design for high availability and redundancy of the APIs and related services that are responsible for provisioning and providing access to storage. It is recommended to design a layer of hardware or software load balancers in order to achieve high availability of the appropriate REST API services to provide uninterrupted service. In some cases, it may also be necessary to deploy an additional layer of load balancing to provide access to back-end database services responsible for servicing and storing the state of block storage volumes. Designing a highly available database solution to store the Block Storage databases is also recommended. A number of highly available database solutions such as Galera and MariaDB can be leveraged to help keep database services online to provide uninterrupted access so that tenants can manage block storage volumes.

In a cloud with extreme demands on block storage the network architecture should take into account the amount of East-West bandwidth that will be required for instances to make use of the available storage resources. Network devices selected should support jumbo frames for transferring large blocks of data. In some cases, it may be necessary to create

an additional back-end storage network dedicated to providing connectivity between instances and block storage resources so that there is no contention of network resources.

# Object Storage fault tolerance and availability

While consistency and partition tolerance are both inherent features of the object storage service, it is important to design the overall storage architecture to ensure that those goals can be met by the system being implemented. The OpenStack object storage service places a specific number of data replicas as objects on resource nodes. These replicas are distributed throughout the cluster based on a consistent hash ring which exists on all nodes in the cluster.

The object storage system should be designed with sufficient number of zones to provide quorum for the number of replicas defined. As an example, with three replicas configured in the Swift cluster, the recommended number of zones to configure within the object storage cluster in order to achieve quorum is 5. While it is possible to deploy a solution with fewer zones, the implied risk of doing so is that some data may not be available and API requests to certain objects stored in the cluster might fail. For this reason, ensure the number of zones in the object storage cluster is properly accounted for.

Each object storage zone should be self-contained within its own availability zone. Each availability zone should have independent access to network, power and cooling infrastructure to ensure uninterrupted access to data. In addition, each availability zone should be serviced by a pool of object storage proxy servers which will provide access to data stored on the object nodes. Object proxies in each region should leverage local read and write affinity so that access to objects is facilitated by local storage resources wherever possible. It is recommended that upstream load balancing be deployed to ensure that proxy services can be distributed across the multiple zones and, in some cases, it may be necessary to make use of third party solutions to aid with geographical distribution of services.

A zone within an object storage cluster is a logical division. A zone can be represented as a disk within a single node, the node itself, a group of nodes in a rack, an entire rack, a data center or even a geographic region. Deciding on the proper zone design is crucial for allowing the object storage cluster to scale while providing an available and redundant storage system. Furthermore, it may be necessary to configure storage policies that have different requirements with regards to replicas, retention and other factors that could heavily affect the design of storage in a specific zone.

# Scaling storage services

Adding storage capacity and bandwidth is a very different process when comparing the block and object storage services. While adding block storage capacity is a relatively simple process, adding capacity and bandwidth to the object storage systems is a complex task that requires careful planning and consideration during the design phase.

# Scaling Block Storage

Block storage pools can be upgraded to add storage capacity rather easily without interruption to the overall block storage service. Nodes can be added to the pool by simply installing and configuring the appropriate hardware and software and then allowing that node to report in to the proper storage pool via the message bus. This is because block storage nodes report into the scheduler service advertising their availability. Once the node is online and available tenants can make use of those storage resources instantly.

In some cases, the demand on block storage from instances may exhaust the available network bandwidth. As a result, design network infrastructure that services block storage resources in such a way that capacity and bandwidth can be added relatively easily. This often involves the use of dynamic routing protocols or advanced networking solutions to allow capacity to be added to downstream devices easily. Both the front-end and back-end storage network designs should encompass the ability to quickly and easily add capacity and bandwidth.

# Scaling Object Storage

Adding back-end storage capacity to an object storage cluster requires careful planning and consideration. In the design phase it is important to determine the maximum partition power required by the object storage service, which determines the maximum number of partitions which can exist. Object storage distributes data among all available storage, but a partition cannot span more than one disk, so the maximum number of partitions can only be as high as the number of disks.

For example, a system that starts with a single disk and a partition power of 3 can have 8 (2^3) partitions. Adding a second disk means that each will (usually) have 4 partitions. The one-disk-per-partition limit means that this system can never have more than 8 disks, limiting its scalability. However, a system that starts with a single disk and a partition power of 10 can have up to 1024 (2^10) disks.

As back-end storage capacity is added to the system, the partition maps cause data to be redistributed amongst storage nodes. In some cases, this replication can consist of extremely large data sets. In these cases, it is recommended to make use of back-end replication links which will not contend with tenants' access to data.

As more tenants begin to access data within the cluster and their data sets grow it will become necessary to add front-end bandwidth to service data access requests. Adding front-end bandwidth to an object storage cluster requires careful planning and design of the object storage proxies that will be used by tenants to gain access to the data, along with the high availability solutions that enable easy scaling of the proxy layer. It is recommended to design a front-end load balancing layer that tenants and consumers use to gain access to data stored within the cluster. This load balancing layer may be distributed across zones, regions or even across geographic boundaries, which may also require that the design encompass geo-location solutions.

In some cases, adding bandwidth and capacity to the network resources servicing requests between proxy servers and storage nodes will be required. For this reason, the network architecture used for access to storage nodes and proxy servers should make use of a design which is scalable.

# Architecture

Storage hardware selection options include three areas:

- Cost
- Performance
- Reliability

The selection of hardware for a storage-focused OpenStack cloud must reflect the fact that the workloads are storage intensive. These workloads are not compute intensive, nor are they consistently network intensive. The network may be heavily utilized to transfer storage, but they are not otherwise network intensive. The hardware selection for a storage-focused OpenStack architecture design must reflect this preference for storage-intensive workloads.

For a storage-focused OpenStack design architecture, the selection of storage hardware will determine the overall performance and scalability of the

design architecture. A number of different factors must be considered in the design process:

**Cost** The overall cost of the solution will play a major role in what storage architecture and the resulting storage hardware that is selected.

**Performance** The performance of the solution, measured by observing the latency of storage I-O requests, also plays a major role. In a compute-focused OpenStack cloud storage latency could potentially be a major consideration, in some compute-intensive workloads, minimizing the delays that the CPU experiences while fetching data from the storage can have a significant impact on the overall performance of the application.

**Scalability** "Scalability" refers to how well the storage solution performs as it is expanded up to its maximum size. A storage solution that performs well in small configurations but has degrading performance as it expands would be considered not scalable. Conversely, a solution that continues to perform well at maximum expansion would be considered scalable. The ability of the storage solution to continue to perform well as it expands is important.

**Expandability** Here we are referring to the overall ability of the solution to grow. A storage solution that expands to 50 PB is considered more expandable than a solution that only scales to 10 PB. Note that this metric is related to but different from scalability which is a measure of the solution's performance as it expands.

Latency is one of the key considerations in a storage-focused OpenStack cloud . Using solid-state disks (SSDs) to minimize latency for instance storage and reduce CPU delays caused by waiting for the storage will have a result of increased performance. It is also recommended to evaluate the gains from using RAID controller cards in compute hosts to improve the performance of the underlying disk subsystem.

The selection of storage architecture (and the corresponding storage hardware, if there is an option) is determined by evaluating possible solutions against the key factors above. This will determine if a scale-out solution (such as Ceph, GlusterFS, or similar) should be used or if a single, highly expandable and scalable centralized storage array would be a better choice.

If a centralized storage array is the right fit for the requirements then the hardware will be determined by the array vendor. It is possible to build a storage array using commodity hardware with Open Source software, but there needs to be access to people with expertise to build such a system. On the other hand, a scale-out storage solution that uses direct-attached storage (DAS) in the servers may be an appropriate choice. If this is true, then the server hardware needs to be configured to support the storage solution.

Some potential impacts that might affect a particular storage architecture (and corresponding storage hardware) of a Storage-focused OpenStack cloud:

**Connectivity** Based on the storage solution selected, ensure the connectivity matches the storage solution requirements. If a centralized storage array is selected it is important to determine how the hypervisors will connect to the storage array. Connectivity can affect latency and thus performance. It is recommended to check that the network characteristics will minimize latency to boost the overall performance of the design.

**Latency** Determine if the use case will have consistent or highly variable latency.

**Throughput** Ensure that the storage solution throughput is optimized based on application requirements.

**Server hardware** Use of DAS impacts the server hardware choice and affects host density, instance density, power density, OS-hypervisor, and management tools, to name a few.

# Compute (server) hardware selection

Compute (server) hardware must be evaluated against four opposing dimensions:

**Server density** A measure of how many servers can fit into a given measure of physical space, such as a rack unit [U].

**Resource capacity** The number of CPU cores, how much RAM, or how much storage a given server will deliver.

| | |
|---|---|
| **Expandability** | The number of additional resources that can be added to a server before it has reached its limit. |
| **Cost** | The relative of the hardware weighted against the level of design effort needed to build the system. |

The dimensions need to be weighed against each other to determine the best design for the desired purpose. For example, increasing server density can mean sacrificing resource capacity or expandability. Increasing resource capacity and expandability can increase cost but decrease server density. Decreasing cost often means decreasing supportability, server density, resource capacity, and expandability.

For a storage-focused OpenStack architecture design, a secondary design consideration for selecting server hardware will be the compute capacity (CPU cores and RAM capacity). As a result, the required server hardware must supply adequate CPU sockets, additional CPU cores, and more RAM; network connectivity and storage capacity are not as critical. The hardware will need to provide enough network connectivity and storage capacity to meet the user requirements, however they are not the primary consideration.

Since there is only a need for adequate CPU and RAM capacity, some server hardware form factors will be better suited to this storage-focused design than others:

- Most blade servers typically support dual-socket multi-core CPUs; to avoid the limit will mean choosing "full width" or "full height" blades, which means losing server density. The high density blade servers (for example, both HP BladeSystem and Dell PowerEdge M1000e), which support up to 16 servers in only 10 rack units using "half height" or "half width" blades, suddenly decrease the density by 50% (only 8 servers in 10 U) if a "full width" or "full height" option is used.

- 1U rack-mounted servers (servers that occupy only a single rack unit) might be able to offer greater server density than a blade server solution (40 servers in a rack, providing space for the top of rack (ToR) switches, versus 32 "full width" or "full height" blade servers in a rack), but often are limited to dual-socket, multi-core CPU configurations. Note that as of the Icehouse release, neither HP, IBM, nor Dell offered 1U rack servers with more than 2 CPU sockets. To obtain greater than dual-socket support in a 1U rack-mount form factor, customers need to buy their systems from Original Design Manufacturers (ODMs) or second-tier manufacturers. This may cause issues for organizations that have preferred

vendor policies or concerns with support and hardware warranties of non-tier 1 vendors.

- 2U rack-mounted servers provide quad-socket, multi-core CPU support but with a corresponding decrease in server density (half the density offered by 1U rack-mounted servers).
- Larger rack-mounted servers, such as 4U servers, often provide even greater CPU capacity. Commonly supporting four or even eight CPU sockets. These servers have greater expandability capacity but such servers have much lower server density and usually greater hardware cost.
- The so-called "sled servers" (rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure) deliver increased density as compared to a typical 1U-2U rack-mounted servers. For example, many sled servers offer four independent dual-socket nodes in 2U for a total of 8 CPU sockets in 2U. However, the dual-socket limitation on individual nodes may not be sufficient to offset their additional cost and configuration complexity.

Other factors that will strongly influence server hardware selection for a storage-focused OpenStack design architecture:

| | |
|---|---|
| **Instance density** | In this architecture, instance density and CPU-RAM oversubscription are lower. More hosts will be required to support the anticipated scale, especially if the design uses dual-socket hardware designs. |
| **Host density** | Another option to address the higher host count is to use a quad socket platform. Taking this approach will decrease host density which also increases rack count. This configuration affects the number of power connections and also impacts network and cooling requirements. |
| **Power and cooling density** | The power and cooling density requirements might be lower than with blade, sled, or 1U server designs due to lower host density (by using 2U, 3U or even 4U server designs). For data centers |

with older infrastructure, this might be a desirable feature.

Storage-focused OpenStack design architecture server hardware selection should focus on a "scale up" versus "scale out" solution. The determination of which will be the best solution, smaller number of larger hosts or a larger number of smaller hosts, will depend of a combination of factors including cost, power, cooling, physical rack and floor space, support-warranty, and manageability.

# Networking hardware selection

Some of the key considerations that should be included in the selection of networking hardware include:

**Port count** The user will require networking hardware that has the requisite port count.

**Port density** The network design will be affected by the physical space that is required to provide the requisite port count. A switch that can provide 48 10 GbE ports in 1U has a much higher port density than a switch that provides 24 10 GbE ports in 2U. On a general scale, a higher port density leaves more rack space for compute or storage components which is preferred. It is also important to consider fault domains and power density. Finally, higher density switches are more expensive, therefore it is important not to over design the network.

**Port speed** The networking hardware must support the proposed network speed, for example: 1 GbE, 10 GbE, or 40 GbE (or even 100 GbE).

**Redundancy** The level of network hardware redundancy required is influenced by the user requirements for high availability and cost considerations. Network redundancy can be achieved by adding redundant power supplies or paired switches. If this is a requirement the hardware will need to support this configuration. User requirements will determine if a completely redundant network infrastructure is required.

**Power requirements** Make sure that the physical data center provides the necessary power for the selected network hardware. This is not typically an issue for top of rack (ToR) switches, but may be an issue for spine switches in a leaf and spine fabric, or end of row (EoR) switches.

**Protocol support** It is possible to gain even more performance out of a single storage system by using specialized network technologies such as RDMA, SRP, iSER and SCST. The specifics for using these technologies is beyond the scope of this book.

# Software selection

Selecting software to be included in a storage-focused OpenStack architecture design includes three areas:

- Operating system (OS) and hypervisor
- OpenStack components
- Supplemental software

Design decisions made in each of these areas impacts the rest of the OpenStack architecture design.

# Operating system and hypervisor

The selection of OS and hypervisor has a significant impact on the overall design and also affects server hardware selection. Ensure that the storage hardware is supported by the selected operating system and hypervisor combination and that the networking hardware selection and topology will work with the chosen operating system and hypervisor combination. For example, if the design uses Link Aggregation Control Protocol (LACP), the OS and hypervisor are both required to support it.

Some areas that could be impacted by the selection of OS and hypervisor include:

**Cost** Selection of a commercially supported hypervisor, such as Microsoft Hyper-V, will result in a different cost model rather than se-

| | |
|---|---|
| | lected a community-supported open source hypervisor like Kinstance or Xen. Similarly, choosing Ubuntu over Red Hat (or vice versa) will have an impact on cost due to support contracts. Conversely, business or application requirements might dictate a specific or commercially supported hypervisor. |
| **Supportability** | Whichever hypervisor is chosen, the staff needs to have appropriate training and knowledge to support the selected OS and hypervisor combination. If they do not training will need to be provided, which could have a cost impact on the design. Another aspect to consider would be the support for the OS-hypervisor. The support of a commercial product such as Red Hat, SUSE, or Windows, is the responsibility of the OS vendor. If an Open Source platform is chosen, the support comes from in-house resources. Either decision has a cost that will have an impact on the design. |
| **Management tools** | The management tools used for Ubuntu and Kinstance differ from the management tools for VMware vSphere. Although both OS and hypervisor combinations are supported by OpenStack, there will naturally be very different impacts to the rest of the design as a result of the selection of one combination versus the other. |
| **Scale and performance** | Make sure that selected OS and hypervisor combination meet the appropriate scale and performance requirements needed for this general purpose OpenStack cloud. The chosen architecture will need to meet the targeted instance-host ratios with the selected OS-hypervisor combination. |
| **Security** | Make sure that the design can accommodate the regular periodic installation of application security patches while maintaining the required workloads. The frequency of |

| | |
|---|---|
| | security patches for the proposed OS-hypervisor combination will have an impact on performance and the patch installation process could affect maintenance windows. |
| **Supported features** | Determine what features of OpenStack are required. This will often determine the selection of the OS-hypervisor combination. Certain features are only available with specific OSs or hypervisors. For example, if certain features are not available, the design might need to be modified to meet the user requirements. |
| **Interoperability** | Consideration should be given to the ability of the selected OS-hypervisor combination to interoperate or co-exist with other OS-hypervisors ,or other software solutions in the overall design, if that is a requirement. Operational and troubleshooting tools for one OS-hypervisor combination may differ from the tools used for another OS-hypervisor combination. As a result, the design will need to address if the two sets of tools need to interoperate. |

# OpenStack components

The selection of OpenStack components has a significant direct impact on the overall design. While there are certain components that will always be present, (Compute and Image Service, for example) there are other services that may not need to be present. As an example, a certain design may not require the Orchestration module. Omitting Orchestration would not typically have a significant impact on the overall design however, if the architecture uses a replacement for OpenStack Object Storage for its storage component, this could potentially have significant impacts on the rest of the design.

A storage-focused design might require the ability to use Orchestration to launch instances with Block Storage volumes to perform storage-intensive processing.

For a storage-focused OpenStack design architecture, the following components would typically be used:

- OpenStack Identity (keystone)
- OpenStack dashboard (horizon)
- OpenStack Compute (nova) (including the use of multiple hypervisor drivers)
- OpenStack Object Storage (swift) (or another object storage solution)
- OpenStack Block Storage (cinder)
- OpenStack Image Service (glance)
- OpenStack Networking (neutron) or legacy networking (nova-network)

The exclusion of certain OpenStack components may limit or constrain the functionality of other components. If a design opts to include Orchestration but exclude Telemetry, then the design will not be able to take advantage of Orchestration's auto scaling functionality (which relies on information from Telemetry). Due to the fact that you can use Orchestration to spin up a large number of instances to perform the compute-intensive processing, including Orchestration in a compute-focused architecture design is strongly recommended.

# Supplemental software

While OpenStack is a fairly complete collection of software projects for building a platform for cloud services, there are additional pieces of software that might need to be added to any given OpenStack design.

# Networking software

OpenStack Networking (neutron) provides a wide variety of networking services for instances. There are many additional networking software packages that may be useful to manage the OpenStack components themselves. Some examples include HAProxy, keepalived, and various routing daemons (like Quagga). Some of these software packages, HAProxy in particular, are described in more detail in the *OpenStack High Availability Guide* (refer to the Network controller cluster stack chapter of the OpenStack High Availability Guide). For a general purpose OpenStack cloud, it is reasonably likely that the OpenStack infrastructure components will need to be highly available, and therefore networking software packages like HAProxy will need to be included.

# Management software

This includes software for providing clustering, logging, monitoring, and alerting. The factors for determining which software packages in this category should be selected is outside the scope of this design guide. This design guide focuses specifically on how the selected supplemental software solution impacts or affects the overall OpenStack cloud design.

Clustering Software, such as Corosync or Pacemaker, is determined primarily by the availability design requirements. Therefore, the impact of including (or not including) these software packages is determined by the availability of the cloud infrastructure and the complexity of supporting the configuration after it is deployed. The *OpenStack High Availability Guide* provides more details on the installation and configuration of Corosync and Pacemaker, should these packages need to be included in the design.

Requirements for logging, monitoring, and alerting are determined by operational considerations. Each of these sub-categories includes a number of various options. For example, in the logging sub-category one might consider Logstash, Splunk, Log Insight, or some other log aggregation-consolidation tool. Logs should be stored in a centralized location to make it easier to perform analytics against the data. Log data analytics engines can also provide automation and issue notification, by providing a mechanism to both alert and automatically attempt to remediate some of the more commonly known issues.

If any of these software packages are needed, then the design must account for the additional resource consumption (CPU, RAM, storage, and network bandwidth for a log aggregation solution, for example). Some other potential design impacts include:

- OS-Hypervisor combination: Ensure that the selected logging, monitoring, or alerting tools support the proposed OS-hypervisor combination.

- Network hardware: The network hardware selection needs to be supported by the logging, monitoring, and alerting software.

# Database software

Virtually all of the OpenStack components require access to back-end database services to store state and configuration information. Choose an appropriate back-end database which will satisfy the availability and fault tolerance requirements of the OpenStack services.

MySQL is generally considered to be the de facto database for OpenStack, however, other compatible databases are also known to work. Note, however, that Telemetry uses MongoDB.

The solution selected to provide high availability for the database will change based on the selected database. If MySQL is selected, then a number of options are available. For active-active clustering a replication technology such as Galera can be used. For active-passive some form of shared storage must be used. Each of these potential solutions has an impact on the design:

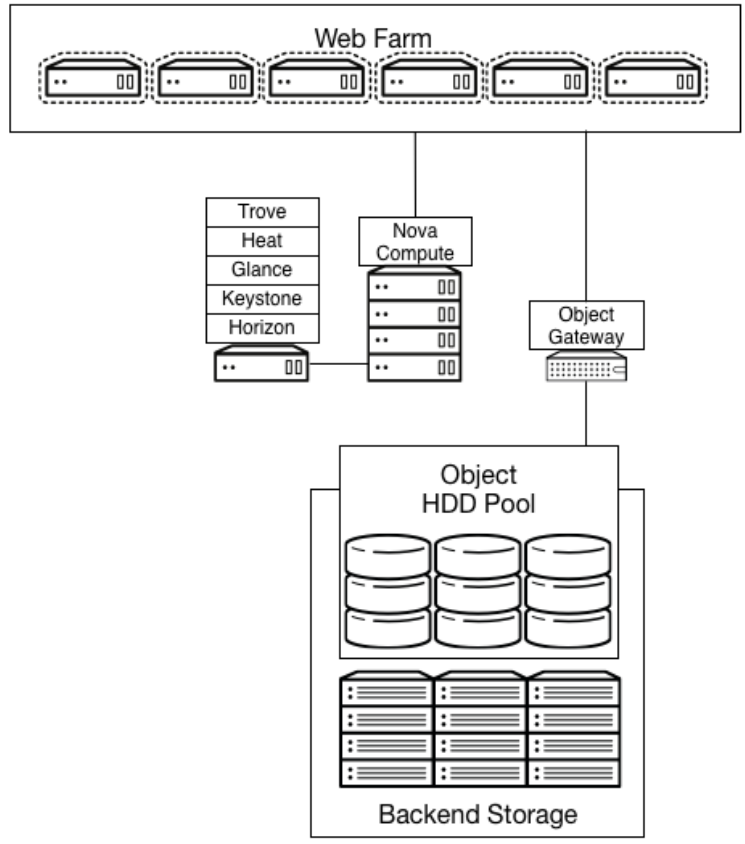
- Solutions that employ Galera/MariaDB will require at least three MySQL nodes.
- MongoDB will have its own design considerations, with regards to making the database highly available.
- OpenStack design, generally, does not include shared storage but for a high availability design some components might require it depending on the specific implementation.

# Prescriptive examples

Storage-focused architectures are highly dependent on the specific use case. Three specific example use cases are discussed in this section: an object store with a RESTful interface, compute analytics with parallel file systems, and a high performance database.

This example describes a REST interface without a high performance requirement, so the presented REST interface does not require a high performance caching tier, and is presented as a traditional Object store running on traditional spindles.

Swift is a highly scalable object store that is part of the OpenStack project. This is a diagram to explain the example architecture:




This example uses the following components:

Network:

- 10 GbE horizontally scalable spine leaf back end storage and front end network.

Storage hardware:

- 10 storage servers each with 12x4 TB disks which equals 480 TB total space with approximately 160 Tb of usable space after replicas.

Proxy:

- 3x proxies
- 2x10 GbE bonded front end

- 2x10 GbE back end bonds
- Approximately 60 Gb of total bandwidth to the back end storage cluster

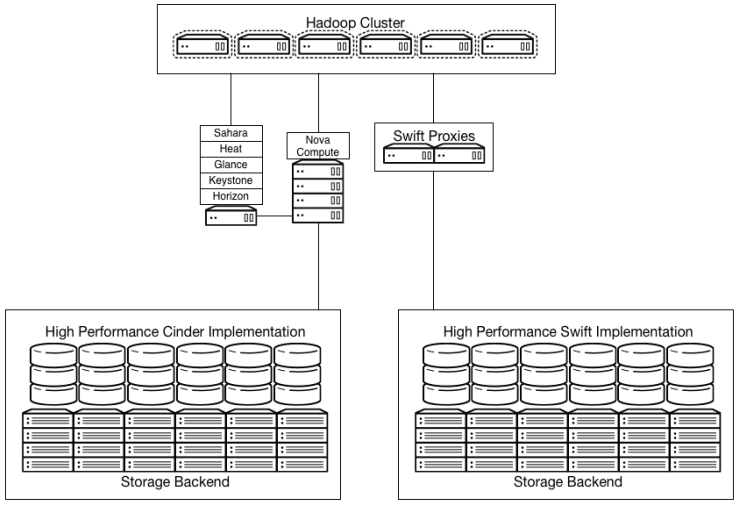
### Note

For some applications, it may be necessary to implement a 3rd-party caching layer to achieve suitable performance.

## Compute analytics with Data processing service for OpenStack

Analytics of large data sets can be highly dependent on the performance of the storage system. Some clouds using storage systems such as HDFS have inefficiencies which can cause performance issues. A potential solution to this is to implement a storage system designed with performance in mind. Traditionally, parallel file systems have filled this need in the HPC space and could be a consideration, when applicable, for large scale performance-oriented systems.

This example discusses an OpenStack Object Store with a high performance requirement. OpenStack has integration with Hadoop through the Data processing project (Sahara), which is leveraged to manage the Hadoop cluster within the cloud.




The actual hardware requirements and configuration are similar to those of the High Performance Database example below. In this case, the archi-

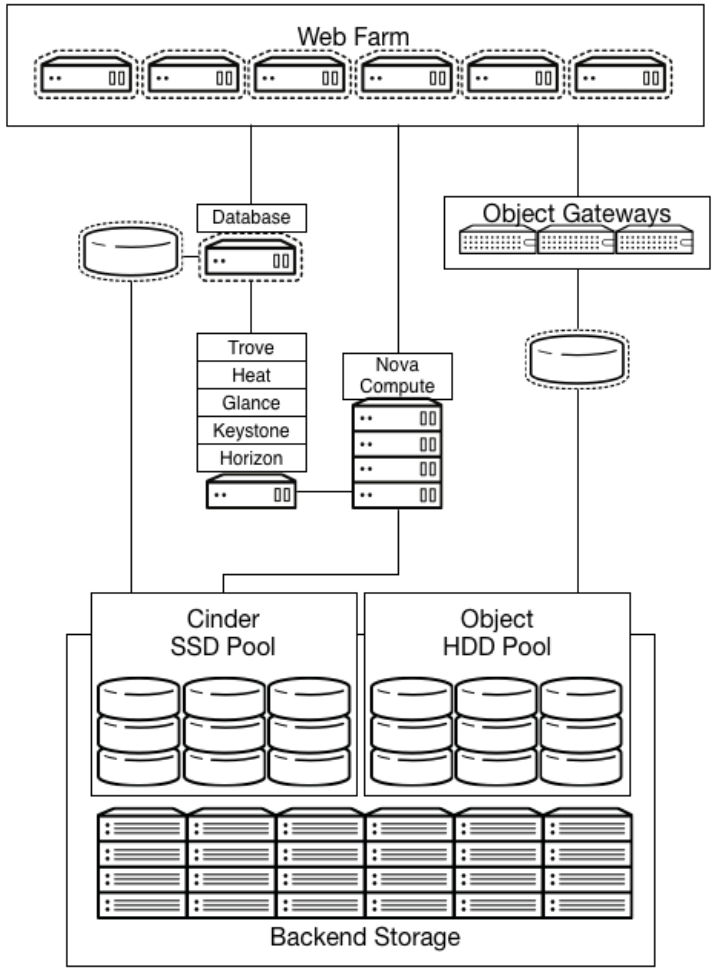
tecture uses Ceph's Swift-compatible REST interface, features that allow for connecting a caching pool to allow for acceleration of the presented pool.

# High performance database with Database service for OpenStack

Databases are a common workload that can greatly benefit from a high performance storage back end. Although enterprise storage is not a requirement, many environments have existing storage that can be used as back ends for an OpenStack cloud. As shown in the following diagram, a storage pool can be carved up to provide block devices with OpenStack Block Storage to instances as well as an object interface. In this example the database I-O requirements were high and demanded storage presented from a fast SSD pool.

A storage system is used to present a LUN that is backed by a set of SSDs using a traditional storage array with OpenStack Block Storage integration or a storage platform such as Ceph or Gluster.

This kind of system can also provide additional performance in other situations. For example, in the database example below, a portion of the SSD pool can act as a block device to the Database server. In the high performance analytics example, the REST interface would be accelerated by the inline SSD cache layer.




Ceph was selected to present a Swift-compatible REST interface, as well as a block level storage from a distributed storage cluster. It is highly flexible and has features that allow to reduce cost of operations such as self healing and auto balancing. Erasure coded pools are used to maximize the amount of usable space. Note that there are special considerations around erasure coded pools, for example, higher computational requirements and limitations on the operations allowed on an object. For example, partial writes are not supported in an erasure coded pool.

A potential architecture for Ceph, as it relates to the examples above, would entail the following:

Network:

- 10 GbE horizontally scalable spine leaf back end storage and front end network

Storage hardware:

- 5 storage servers for caching layer 24x1 TB SSD
- 10 storage servers each with 12x4 TB disks which equals 480 TB total space with about approximately 160 Tb of usable space after 3 replicas

REST proxy:

- 3x proxies
- 2x10 GbE bonded front end
- 2x10 GbE back end bonds
- Approximately 60 Gb of total bandwidth to the back end storage cluster

The SSD cache layer is used to present block devices directly to Hypervisors or instances. The SSD cache systems can also be used as an inline cache for the REST interface.

# 5. Network focused

## Table of Contents



All OpenStack deployments are dependent, to some extent, on network communication in order to function properly due to a service-based nature. In some cases, however, use cases dictate that the network is elevated beyond simple infrastructure. This chapter is a discussion of architectures that are more reliant or focused on network services. These architectures are heavily dependent on the network infrastructure and need to be architected so that the network services perform and are reliable in order to satisfy user and application requirements.

Some possible use cases include:

**Content delivery network**

This could include streaming video, photographs or any other cloud based repository of data that is distributed to a large number of end users. Mass market streaming video will be very heavily affected by the network configurations that would affect latency, bandwidth, and the distribution of instances. Not all video streaming is consumer focused. For example, multicast videos (used for media, press conferences, corporate presentations, web conferencing services, and so on) can also utilize a content delivery network. Content delivery will be affected by the location of the video repository and its relationship to end users. Performance is also affected by network throughput of the backend systems, as well as

| | |
|---|---|
| | the WAN architecture and the cache methodology. |
| **Network management functions** | A cloud that provides network service functions would be built to support the delivery of back-end network services such as DNS, NTP or SNMP and would be used by a company for internal network management. |
| **Network service offerings** | A cloud can be used to run customer facing network tools to support services. For example, VPNs, MPLS private networks, GRE tunnels and others. |
| **Web portals or web services** | Web servers are a common application for cloud services and it is recommended to have an understanding of the network requirements. The network will need to be able to scale out to meet user demand and deliver webpages with a minimum of latency. Internal east-west and north-south network bandwidth must be considered depending on the details of the portal architecture. |
| **High speed and high volume transactional systems** | These types of applications are very sensitive to network configurations. Examples include many financial systems, credit card transaction applications, trading and other extremely high volume systems. These systems are sensitive to network jitter and latency. They also have a high volume of both east-west and north-south network traffic that needs to be balanced to maximize efficiency of the data delivery. Many of these systems have large high performance database back ends that need to be accessed. |
| **High availability** | These types of use cases are highly dependent on the proper sizing of the |

| | |
|---|---|
| | network to maintain replication of data between sites for high availability. If one site becomes unavailable, the extra sites will be able to serve the displaced load until the original site returns to service. It is important to size network capacity to handle the loads that are desired. |
| **Big data** | Clouds that will be used for the management and collection of big data (data ingest) will have a significant demand on network resources. Big data often uses partial replicas of the data to maintain data integrity over large distributed clouds. Other big data applications that require a large amount of network resources are Hadoop, Cassandra, NuoDB, RIAK and other No-SQL and distributed databases. |
| **Virtual desktop infrastructure (VDI)** | This use case is very sensitive to network congestion, latency, jitter and other network characteristics. Like video streaming, the user experience is very important however, unlike video streaming, caching is not an option to offset the network issues. VDI requires both upstream and downstream traffic and cannot rely on caching for the delivery of the application to the end user. |
| **Voice over IP (VoIP)** | This is extremely sensitive to network congestion, latency, jitter and other network characteristics. VoIP has a symmetrical traffic pattern and it requires network quality of service (QoS) for best performance. It may also require an active queue management implementation to ensure delivery. Users are very sensitive to latency and jitter fluctuations and can detect them at very low levels. |

| | |
|---|---|
| **Video Conference or web conference** | This also is extremely sensitive to network congestion, latency, jitter and other network flaws. Video Conferencing has a symmetrical traffic pattern, but unless the network is on an MPLS private network, it cannot use network quality of service (QoS) to improve performance. Similar to VOIP, users will be sensitive to network performance issues even at low levels. |
| **High performance computing (HPC)** | This is a complex use case that requires careful consideration of the traffic flows and usage patterns to address the needs of cloud clusters. It has high East-West traffic patterns for distributed computing, but there can be substantial North-South traffic depending on the specific application. |

# User requirements

Network focused architectures vary from the general purpose designs. They are heavily influenced by a specific subset of applications that interact with the network in a more impacting way. Some of the business requirements that will influence the design include:

- User experience: User experience is impacted by network latency through slow page loads, degraded video streams, and low quality VoIP sessions. Users are often not aware of how network design and architecture affects their experiences. Both enterprise customers and end-users rely on the network for delivery of an application. Network performance problems can provide a negative experience for the end-user, as well as productivity and economic loss.

- Regulatory requirements: Networks need to take into consideration any regulatory requirements about the physical location of data as it traverses the network. For example, Canadian medical records cannot pass outside of Canadian sovereign territory. Another network consideration is maintaining network segregation of private data flows and ensuring that the network between cloud locations is encrypted where required. Network architectures are affected by regulatory requirements for encryption and protection of data in flight as the data moves through various networks.

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.
- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.
- Data compliance policies governing where information needs to reside in certain locations due to regular issues and, more importantly, where it cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union (http://ec.europa.eu/justice/data-protection/) and the requirements of the Financial Industry Regulatory Authority (http://www.finra.org/Industry/Regulation/FINRARules) in the United States. Consult a local regulatory body for more information.

# High availability issues

OpenStack installations with high demand on network resources have high availability requirements that are determined by the application and use case. Financial transaction systems will have a much higher requirement for high availability than a development application. Forms of network availability, for example quality of service (QoS), can be used to improve the network performance of sensitive applications, for example VoIP and video streaming.

Often, high performance systems will have SLA requirements for a minimum QoS with regard to guaranteed uptime, latency and bandwidth. The level of the SLA can have a significant impact on the network architecture and requirements for redundancy in the systems.

# Risks

| | |
|---|---|
| **Network misconfigurations** | Configuring incorrect IP addresses, VLANs, and routes can cause outages to areas of the network or, in the |

| | |
|---|---|
| | worst-case scenario, the entire cloud infrastructure. Misconfigurations can cause disruptive problems and should be automated to minimize the opportunity for operator error. |
| **Capacity planning** | Cloud networks need to be managed for capacity and growth over time. There is a risk that the network will not grow to support the workload. Capacity planning includes the purchase of network circuits and hardware that can potentially have lead times measured in months or more. |
| **Network tuning** | Cloud networks need to be configured to minimize link loss, packet loss, packet storms, broadcast storms, and loops. |
| **Single Point Of Failure (SPOF)** | High availability must be taken into account even at the physical and environmental layers. If there is a single point of failure due to only one upstream link, or only one power supply, an outage becomes unavoidable. |
| **Complexity** | An overly complex network design becomes difficult to maintain and troubleshoot. While automated tools that handle overlay networks or device level configuration can mitigate this, non-traditional interconnects between functions and specialized hardware need to be well documented or avoided to prevent outages. |
| **Non-standard features** | There are additional risks that arise from configuring the cloud network to take advantage of vendor specific features. One example is multi-link aggregation (MLAG) that is being used to provide redundancy at the aggregator switch level of the network. MLAG is not a standard and, as a result, each vendor has their own proprietary im- |

plementation of the feature. MLAG architectures are not interoperable across switch vendors, which leads to vendor lock-in, and can cause delays or inability when upgrading components.

## Security

Security is often overlooked or added after a design has been implemented. Consider security implications and requirements before designing the physical and logical network topologies. Some of the factors that need to be addressed include making sure the networks are properly segregated and traffic flows are going to the correct destinations without crossing through locations that are undesirable. Some examples of factors that need to be taken into consideration are:

- Firewalls
- Overlay interconnects for joining separated tenant networks
- Routing through or avoiding specific networks

Another security vulnerability that must be taken into account is how networks are attached to hypervisors. If a network must be separated from other systems at all costs, it may be necessary to schedule instances for that network onto dedicated compute nodes. This may also be done to mitigate against exploiting a hypervisor breakout allowing the attacker access to networks from a compromised instance.

# Technical considerations

When you design an OpenStack network architecture, you must consider layer-2 and layer-3 issues. Layer-2 decisions involve those made at the data-link layer, such as the decision to use Ethernet versus Token Ring. Layer-3 decisions involve those made about the protocol layer and the point when IP comes into the picture. As an example, a completely internal OpenStack network can exist at layer 2 and ignore layer 3 however, in order for any traffic to go outside of that cloud, to another network, or to the Internet, a layer-3 router or switch must be involved.

The past few years have seen two competing trends in networking. One trend leans towards building data center network architectures based on layer-2 networking. Another trend treats the cloud environment essential-

ly as a miniature version of the Internet. This approach is radically different from the network architecture approach that is used in the staging environment: the Internet is based entirely on layer-3 routing rather than layer-2 switching.

A network designed on layer-2 protocols has advantages over one designed on layer-3 protocols. In spite of the difficulties of using a bridge to perform the network role of a router, many vendors, customers, and service providers choose to use Ethernet in as many parts of their networks as possible. The benefits of selecting a layer-2 design are:

- Ethernet frames contain all the essentials for networking. These include, but are not limited to, globally unique source addresses, globally unique destination addresses, and error control.
- Ethernet frames can carry any kind of packet. Networking at layer 2 is independent of the layer-3 protocol.
- More layers added to the Ethernet frame only slow the networking process down. This is known as 'nodal processing delay'.
- Adjunct networking features, for example class of service (CoS) or multicasting, can be added to Ethernet as readily as IP networks.
- VLANs are an easy mechanism for isolating networks.

Most information starts and ends inside Ethernet frames. Today this applies to data, voice (for example, VoIP) and video (for example, web cameras). The concept is that, if more of the end-to-end transfer of information from a source to a destination can be done in the form of Ethernet frames, more of the benefits of Ethernet can be realized on the network. Though it is not a substitute for IP networking, networking at layer 2 can be a powerful adjunct to IP networking.

Layer-2 Ethernet usage has these advantages over layer-3 IP network usage:

- Speed
- Reduced overhead of the IP hierarchy.
- No need to keep track of address configuration as systems are moved around. Whereas the simplicity of layer-2 protocols might work well in a data center with hundreds of physical machines, cloud data centers have the additional burden of needing to keep track of all virtual machine ad-

dresses and networks. In these data centers, it is not uncommon for one physical node to support 30-40 instances.

### Important

Networking at the frame level says nothing about the presence or absence of IP addresses at the packet level. Almost all ports, links, and devices on a network of LAN switches still have IP addresses, as do all the source and destination hosts. There are many reasons for the continued need for IP addressing. The largest one is the need to manage the network. A device or link without an IP address is usually invisible to most management applications. Utilities including remote access for diagnostics, file transfer of configurations and software, and similar applications cannot run without IP addresses as well as MAC addresses.

## Layer-2 architecture limitations

Outside of the traditional data center the limitations of layer-2 network architectures become more obvious.

- Number of VLANs is limited to 4096.
- The number of MACs stored in switch tables is limited.
- The need to maintain a set of layer-4 devices to handle traffic control must be accommodated.
- MLAG, often used for switch redundancy, is a proprietary solution that does not scale beyond two devices and forces vendor lock-in.
- It can be difficult to troubleshoot a network without IP addresses and ICMP.
- Configuring *ARP* is considered complicated on large layer-2 networks.
- All network devices need to be aware of all MACs, even instance MACs, so there is constant churn in MAC tables and network state changes as instances are started or stopped.
- Migrating MACs (instance migration) to different physical locations are a potential problem if ARP table timeouts are not set properly.

It is important to know that layer 2 has a very limited set of network management tools. It is very difficult to control traffic, as it does not have

mechanisms to manage the network or shape the traffic, and network troubleshooting is very difficult. One reason for this difficulty is network devices have no IP addresses. As a result, there is no reasonable way to check network delay in a layer-2 network.

On large layer-2 networks, configuring ARP learning can also be complicated. The setting for the MAC address timer on switches is critical and, if set incorrectly, can cause significant performance problems. As an example, the Cisco default MAC address timer is extremely long. Migrating MACs to different physical locations to support instance migration can be a significant problem. In this case, the network information maintained in the switches could be out of sync with the new location of the instance.

In a layer-2 network, all devices are aware of all MACs, even those that belong to instances. The network state information in the backbone changes whenever an instance is started or stopped. As a result there is far too much churn in the MAC tables on the backbone switches.

## Layer-3 architecture advantages

In the layer 3 case, there is no churn in the routing tables due to instances starting and stopping. The only time there would be a routing state change would be in the case of a Top of Rack (ToR) switch failure or a link failure in the backbone itself. Other advantages of using a layer-3 architecture include:

- Layer-3 networks provide the same level of resiliency and scalability as the Internet.
- Controlling traffic with routing metrics is straightforward.
- Layer 3 can be configured to use *BGP* confederation for scalability so core routers have state proportional to the number of racks, not to the number of servers or instances.
- Routing ensures that instance MAC and IP addresses out of the network core reducing state churn. Routing state changes only occur in the case of a ToR switch failure or backbone link failure.
- There are a variety of well tested tools, for example ICMP, to monitor and manage traffic.
- Layer-3 architectures allow for the use of Quality of Service (QoS) to manage network performance.

## Layer-3 architecture limitations

The main limitation of layer 3 is that there is no built-in isolation mechanism comparable to the VLANs in layer-2 networks. Furthermore, the hierarchical nature of IP addresses means that an instance will also be on the same subnet as its physical host. This means that it cannot be migrated outside of the subnet easily. For these reasons, network virtualization needs to use IP *encapsulation* and software at the end hosts for both isolation, as well as for separation of the addressing in the virtual layer from addressing in the physical layer. Other potential disadvantages of layer 3 include the need to design an IP addressing scheme rather than relying on the switches to automatically keep track of the MAC addresses and to configure the interior gateway routing protocol in the switches.

# Network recommendations overview

OpenStack has complex networking requirements for several reasons. Many components interact at different levels of the system stack that adds complexity. Data flows are complex. Data in an OpenStack cloud moves both between instances across the network (also known as East-West), as well as in and out of the system (also known as North-South). Physical server nodes have network requirements that are independent of those used by instances which need to be isolated from the core network to account for scalability. It is also recommended to functionally separate the networks for security purposes and tune performance through traffic shaping.

A number of important general technical and business factors need to be taken into consideration when planning and designing an OpenStack network. They include:

- A requirement for vendor independence. To avoid hardware or software vendor lock-in, the design should not rely on specific features of a vendor's router or switch.
- A requirement to massively scale the ecosystem to support millions of end users.
- A requirement to support indeterminate platforms and applications.
- A requirement to design for cost efficient operations to take advantage of massive scale.
- A requirement to ensure that there is no single point of failure in the cloud ecosystem.

- A requirement for high availability architecture to meet customer SLA requirements.
- A requirement to be tolerant of rack level failure.
- A requirement to maximize flexibility to architect future production environments.

Keeping all of these in mind, the following network design recommendations can be made:

- Layer-3 designs are preferred over layer-2 architectures.
- Design a dense multi-path network core to support multi-directional scaling and flexibility.
- Use hierarchical addressing because it is the only viable option to scale network ecosystem.
- Use virtual networking to isolate instance service network traffic from the management and internal network traffic.
- Isolate virtual networks using encapsulation technologies.
- Use traffic shaping for performance tuning.
- Use eBGP to connect to the Internet up-link.
- Use iBGP to flatten the internal traffic on the layer-3 mesh.
- Determine the most effective configuration for block storage network.

# Additional considerations

There are numerous topics to consider when designing a network-focused OpenStack cloud.

## OpenStack Networking versus legacy networking (nova-network) considerations

Selecting the type of networking technology to implement depends on many factors. OpenStack Networking (neutron) and legacy networking (nova-network) both have their advantages and disadvantages. They are

both valid and supported options that fit different use cases as described in the following table.

| Legacy networking (nova-network) | OpenStack Networking |
|---|---|
| Simple, single agent | Complex, multiple agents |
| More mature, established | Newer, maturing |
| Flat or VLAN | Flat, VLAN, Overlays, L2-L3, SDN |
| No plug-in support | Plug-in support for 3rd parties |
| Scales well | Scaling requires 3rd party plug-ins |
| No multi-tier topologies | Multi-tier topologies |

## Redundant networking: ToR switch high availability risk analysis

A technical consideration of networking is the idea that switching gear in the data center that should be installed with backup switches in case of hardware failure.

Research into the mean time between failures (MTBF) on switches is between 100,000 and 200,000 hours. This number is dependent on the ambient temperature of the switch in the data center. When properly cooled and maintained, this translates to between 11 and 22 years before failure. Even in the worst case of poor ventilation and high ambient temperatures in the data center, the MTBF is still 2-3 years. This is based on published research found at http://www.garrettcom.com/techsupport/papers/ethernet_switch_reliability.pdf and http://www.n-tron.com/pdf/network_availability.pdf.

In most cases, it is much more economical to only use a single switch with a small pool of spare switches to replace failed units than it is to outfit an entire data center with redundant switches. Applications should also be able to tolerate rack level outages without affecting normal operations since network and compute resources are easily provisioned and plentiful.

## Preparing for the future: IPv6 support

One of the most important networking topics today is the impending exhaustion of IPv4 addresses. In early 2014, ICANN announced that they started allocating the final IPv4 address blocks to the Regional Internet Registries (http://www.internetsociety.org/deploy360/blog/2014/05/goodbye-ipv4-iana-starts-allocating-final-address-blocks/). This means the IPv4 address space is close to being fully allocated. As a result, it will soon

become difficult to allocate more IPv4 addresses to an application that has experienced growth, or is expected to scale out, due to the lack of unallocated IPv4 address blocks.

For network focused applications the future is the IPv6 protocol. IPv6 increases the address space significantly, fixes long standing issues in the IPv4 protocol, and will become an essential for network focused applications in the future.

OpenStack Networking supports IPv6 when configured to take advantage of the feature. To enable it, simply create an IPv6 subnet in Networking and use IPv6 prefixes when creating security groups.

## Asymmetric links

When designing a network architecture, the traffic patterns of an application will heavily influence the allocation of total bandwidth and the number of links that are used to send and receive traffic. Applications that provide file storage for customers will allocate bandwidth and links to favor incoming traffic, whereas video streaming applications will allocate bandwidth and links to favor outgoing traffic.

## Performance

It is important to analyze the applications' tolerance for latency and jitter when designing an environment to support network focused applications. Certain applications, for example VoIP, are less tolerant of latency and jitter. Where latency and jitter are concerned, certain applications may require tuning of QoS parameters and network device queues to ensure that they are queued for transmit immediately or guaranteed minimum bandwidth. Since OpenStack currently does not support these functions, some considerations may need to be made for the network plug-in selected.

The location of a service may also impact the application or consumer experience. If an application is designed to serve differing content to differing users it will need to be designed to properly direct connections to those specific locations. Use a multi-site installation for these situations, where appropriate.

Networking can be implemented in two separate ways. The legacy networking (nova-network) provides a flat DHCP network with a single broadcast domain. This implementation does not support tenant isolation networks or advanced plug-ins, but it is currently the only way to implement a distributed layer-3 agent using the multi_host configuration. OpenStack

Networking (neutron) is the official networking implementation and provides a pluggable architecture that supports a large variety of network methods. Some of these include a layer-2 only provider network model, external device plug-ins, or even OpenFlow controllers.

Networking at large scales becomes a set of boundary questions. The determination of how large a layer-2 domain needs to be is based on the amount of nodes within the domain and the amount of broadcast traffic that passes between instances. Breaking layer-2 boundaries may require the implementation of overlay networks and tunnels. This decision is a balancing act between the need for a smaller overhead or a need for a smaller domain.

When selecting network devices, be aware that making this decision based on largest port density often comes with a drawback. Aggregation switches and routers have not all kept pace with Top of Rack switches and may induce bottlenecks on north-south traffic. As a result, it may be possible for massive amounts of downstream network utilization to impact upstream network devices, impacting service to the cloud. Since OpenStack does not currently provide a mechanism for traffic shaping or rate limiting, it is necessary to implement these features at the network hardware level.

# Operational considerations

Network focused OpenStack clouds have a number of operational considerations that will influence the selected design. Topics including, but not limited to, dynamic routing of static routes, service level agreements, and ownership of user management all need to be considered.

One of the first required decisions is the selection of a telecom company or transit provider. This is especially true if the network requirements include external or site-to-site network connectivity.

Additional design decisions need to be made about monitoring and alarming. These can be an internal responsibility or the responsibility of the external provider. In the case of using an external provider, SLAs will likely apply. In addition, other operational considerations such as bandwidth, latency, and jitter can be part of a service level agreement.

The ability to upgrade the infrastructure is another subject for consideration. As demand for network resources increase, operators will be required to add additional IP address blocks and add additional bandwidth capacity. Managing hardware and software life cycle events, for example up-

grades, decommissioning, and outages while avoiding service interruptions for tenants, will also need to be considered.

Maintainability will also need to be factored into the overall network design. This includes the ability to manage and maintain IP addresses as well as the use of overlay identifiers including VLAN tag IDs, GRE tunnel IDs, and MPLS tags. As an example, if all of the IP addresses have to be changed on a network, a process known as renumbering, then the design needs to support the ability to do so.

Network focused applications themselves need to be addressed when concerning certain operational realities. For example, the impending exhaustion of IPv4 addresses, the migration to IPv6 and the utilization of private networks to segregate different types of traffic that an application receives or generates. In the case of IPv4 to IPv6 migrations, applications should follow best practices for storing IP addresses. It is further recommended to avoid relying on IPv4 features that were not carried over to the IPv6 protocol or have differences in implementation.

When using private networks to segregate traffic, applications should create private tenant networks for database and data storage network traffic, and utilize public networks for client-facing traffic. By segregating this traffic, quality of service and security decisions can be made to ensure that each network has the correct level of service that it requires.

Finally, decisions must be made about the routing of network traffic. For some applications, a more complex policy framework for routing must be developed. The economic cost of transmitting traffic over expensive links versus cheaper links, in addition to bandwidth, latency, and jitter requirements, can be used to create a routing policy that will satisfy business requirements.

How to respond to network events must also be taken into consideration. As an example, how load is transferred from one link to another during a failure scenario could be a factor in the design. If network capacity is not planned correctly, failover traffic could overwhelm other ports or network links and create a cascading failure scenario. In this case, traffic that fails over to one link overwhelms that link and then moves to the subsequent links until the all network traffic stops.

# Architecture

Network focused OpenStack architectures have many similarities to other OpenStack architecture use cases. There a number of very specific con-

siderations to keep in mind when designing for a network-centric or network-heavy application environment.

Networks exist to serve a as medium of transporting data between systems. It is inevitable that an OpenStack design have inter-dependencies with non-network portions of OpenStack as well as on external systems. Depending on the specific workload, there may be major interactions with storage systems both within and external to the OpenStack environment. For example, if the workload is a content delivery network, then the interactions with storage will be two-fold. There will be traffic flowing to and from the storage array for ingesting and serving content in a north-south direction. In addition, there is replication traffic flowing in an east-west direction.

Compute-heavy workloads may also induce interactions with the network. Some high performance compute applications require network-based memory mapping and data sharing and, as a result, will induce a higher network load when they transfer results and data sets. Others may be highly transactional and issue transaction locks, perform their functions and rescind transaction locks at very high rates. This also has an impact on the network performance.

Some network dependencies are going to be external to OpenStack. While OpenStack Networking is capable of providing network ports, IP addresses, some level of routing, and overlay networks, there are some other functions that it cannot provide. For many of these, external systems or equipment may be required to fill in the functional gaps. Hardware load balancers are an example of equipment that may be necessary to distribute workloads or offload certain functions. Note that, as of the Icehouse release, dynamic routing is currently in its infancy within OpenStack and may need to be implemented either by an external device or a specialized service instance within OpenStack. Tunneling is a feature provided by OpenStack Networking, however it is constrained to a Networking-managed region. If the need arises to extend a tunnel beyond the OpenStack region to either another region or an external system, it is necessary to implement the tunnel itself outside OpenStack or by using a tunnel management system to map the tunnel or overlay to an external tunnel. OpenStack does not currently provide quotas for network resources. Where network quotas are required, it is necessary to implement quality of service management outside of OpenStack. In many of these instances, similar solutions for traffic shaping or other network functions will be needed.

Depending on the selected design, Networking itself might not even support the required *layer-3 network* functionality. If you choose to use the

provider networking mode without running the layer-3 agent, you must install an external router to provide layer-3 connectivity to outside systems.

Interaction with orchestration services is inevitable in larger-scale deployments. The Orchestration module is capable of allocating network resource defined in templates to map to tenant networks and for port creation, as well as allocating floating IPs. If there is a requirement to define and manage network resources in using orchestration, it is recommended that the design include the Orchestration module to meet the demands of users.

# Design impacts

A wide variety of factors can affect a network focused OpenStack architecture. While there are some considerations shared with a general use case, specific workloads related to network requirements will influence network design decisions.

One decision includes whether or not to use Network Address Translation (NAT) and where to implement it. If there is a requirement for floating IPs to be available instead of using public fixed addresses then NAT is required. This can be seen in network management applications that rely on an IP endpoint. An example of this is a DHCP relay that needs to know the IP of the actual DHCP server. In these cases it is easier to automate the infrastructure to apply the target IP to a new instance rather than reconfigure legacy or external systems for each new instance.

NAT for floating IPs managed by Networking will reside within the hypervisor but there are also versions of NAT that may be running elsewhere. If there is a shortage of IPv4 addresses there are two common methods to mitigate this externally to OpenStack. The first is to run a load balancer either within OpenStack as an instance, or use an external load balancing solution. In the internal scenario, load balancing software, such as HAproxy, can be managed with Networking's Load-Balancer-as-a-Service (LBaaS). This is specifically to manage the Virtual IP (VIP) while a dual-homed connection from the HAproxy instance connects the public network with the tenant private network that hosts all of the content servers. In the external scenario, a load balancer would need to serve the VIP and also be joined to the tenant overlay network through external means or routed to it via private addresses.

Another kind of NAT that may be useful is protocol NAT. In some cases it may be desirable to use only IPv6 addresses on instances and operate either an instance or an external service to provide a NAT-based transition

technology such as NAT64 and DNS64. This provides the ability to have a globally routable IPv6 address while only consuming IPv4 addresses as necessary or in a shared manner.

Application workloads will affect the design of the underlying network architecture. If a workload requires network-level redundancy, the routing and switching architecture will have to accommodate this. There are differing methods for providing this that are dependent on the network hardware selected, the performance of the hardware, and which networking model is deployed. Some examples of this are the use of Link aggregation (LAG) or Hot Standby Router Protocol (HSRP). There are also the considerations of whether to deploy OpenStack Networking or legacy networking (nova-network) and which plug-in to select for OpenStack Networking. If using an external system, Networking will need to be configured to run *layer 2* with a provider network configuration. For example, it may be necessary to implement HSRP to terminate layer-3 connectivity.

Depending on the workload, overlay networks may or may not be a recommended configuration. Where application network connections are small, short lived or bursty, running a dynamic overlay can generate as much bandwidth as the packets it carries. It also can induce enough latency to cause issues with certain applications. There is an impact to the device generating the overlay which, in most installations, will be the hypervisor. This will cause performance degradation on packet per second and connection per second rates.

Overlays also come with a secondary option that may or may not be appropriate to a specific workload. While all of them will operate in full mesh by default, there might be good reasons to disable this function because it may cause excessive overhead for some workloads. Conversely, other workloads will operate without issue. For example, most web services applications will not have major issues with a full mesh overlay network, while some network monitoring tools or storage replication workloads will have performance issues with throughput or excessive broadcast traffic.

Many people overlook an important design decision: The choice of layer-3 protocols. While OpenStack was initially built with only IPv4 support, Networking now supports IPv6 and dual-stacked networks. Note that, as of the Icehouse release, this only includes stateless address autoconfiguration but work is in progress to support stateless and stateful DHCPv6 as well as IPv6 floating IPs without NAT. Some workloads become possible through the use of IPv6 and IPv6 to IPv4 reverse transition mechanisms such as NAT64 and DNS64 or *6to4*, because these options are available. This will alter the requirements for any address plan as single-stacked and transitional IPv6 deployments can alleviate the need for IPv4 addresses.

As of the Icehouse release, OpenStack has limited support for dynamic routing, however there are a number of options available by incorporating third party solutions to implement routing within the cloud including network equipment, hardware nodes, and instances. Some workloads will perform well with nothing more than static routes and default gateways configured at the layer-3 termination point. In most cases this will suffice, however some cases require the addition of at least one type of dynamic routing protocol if not multiple protocols. Having a form of interior gateway protocol (IGP) available to the instances inside an OpenStack installation opens up the possibility of use cases for anycast route injection for services that need to use it as a geographic location or failover mechanism. Other applications may wish to directly participate in a routing protocol, either as a passive observer as in the case of a looking glass, or as an active participant in the form of a route reflector. Since an instance might have a large amount of compute and memory resources, it is trivial to hold an entire unpartitioned routing table and use it to provide services such as network path visibility to other applications or as a monitoring tool.

Path maximum transmission unit (MTU) failures are lesser known but harder to diagnose. The MTU must be large enough to handle normal traffic, overhead from an overlay network, and the desired layer-3 protocol. When you add externally built tunnels, the MTU packet size is reduced. In this case, you must pay attention to the fully calculated MTU size because some systems are configured to ignore or drop path MTU discovery packets.

# Tunable networking components

Consider configurable networking components related to an OpenStack architecture design when designing for network intensive workloads include MTU and QoS. Some workloads will require a larger MTU than normal based on a requirement to transfer large blocks of data. When providing network service for applications such as video streaming or storage replication, it is recommended to ensure that both OpenStack hardware nodes and the supporting network equipment are configured for jumbo frames where possible. This will allow for a better utilization of available bandwidth. Configuration of jumbo frames should be done across the complete path the packets will traverse. If one network component is not capable of handling jumbo frames then the entire path will revert to the default MTU.

Quality of Service (QoS) also has a great impact on network intensive workloads by providing instant service to packets which have a higher priority due to their ability to be impacted by poor network performance. In

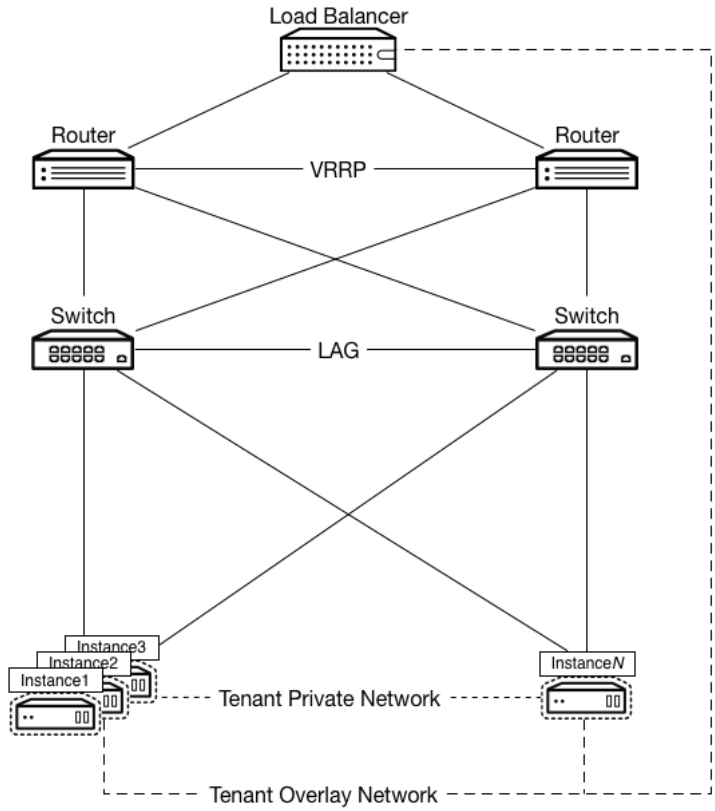
applications such as Voice over IP (VoIP) differentiated services code points are a near requirement for proper operation. QoS can also be used in the opposite direction for mixed workloads to prevent low priority but high bandwidth applications, for example backup services, video conferencing or file sharing, from blocking bandwidth that is needed for the proper operation of other workloads. It is possible to tag file storage traffic as a lower class, such as best effort or scavenger, to allow the higher priority traffic through. In cases where regions within a cloud might be geographically distributed it may also be necessary to plan accordingly to implement WAN optimization to combat latency or packet loss.

# Prescriptive examples

A large-scale web application has been designed with cloud principles in mind. The application is designed to scale horizontally in a bursting fashion and will generate a high instance count. The application requires an SSL connection to secure data and must not lose connection state to individual servers.

An example design for this workload is depicted in the figure below. In this example, a hardware load balancer is configured to provide SSL offload functionality and to connect to tenant networks in order to reduce address consumption. This load balancer is linked to the routing architecture as it will service the VIP for the application. The router and load balancer are configured with GRE tunnel ID of the application's tenant network and provided an IP address within the tenant subnet but outside of the address pool. This is to ensure that the load balancer can communicate with the application's HTTP servers without requiring the consumption of a public IP address.

Because sessions persist until they are closed, the routing and switching architecture is designed for high availability. Switches are meshed to each hypervisor and each other, and also provide an MLAG implementation to ensure that layer-2 connectivity does not fail. Routers are configured with VRRP and fully meshed with switches to ensure layer-3 connectivity. Since GRE is used as an overlay network, Networking is installed and configured to use the Open vSwitch agent in GRE tunnel mode. This ensures all devices can reach all other devices and that tenant networks can be created for private addressing links to the load balancer.




A web service architecture has many options and optional components. Due to this, it can fit into a large number of other OpenStack designs however a few key components will need to be in place to handle the nature of most web-scale workloads. The user needs the following components:

- OpenStack Controller services (Image, Identity, Networking and supporting services such as MariaDB and RabbitMQ)
- OpenStack Compute running KVM hypervisor
- OpenStack Object Storage
- Orchestration module
- Telemetry module

Beyond the normal Identity, Compute, Image Service and Object Storage components, the Orchestration module is a recommended component to

handle properly scaling the workloads to adjust to demand. Due to the requirement for auto-scaling, the design includes the Telemetry module. Web services tend to be bursty in load, have very defined peak and valley usage patterns and, as a result, benefit from automatic scaling of instances based upon traffic. At a network level, a split network configuration will work well with databases residing on private tenant networks since these do not emit a large quantity of broadcast traffic and may need to interconnect to some databases for content.

## Load balancing

Load balancing was included in this design to spread requests across multiple instances. This workload scales well horizontally across large numbers of instances. This allows instances to run without publicly routed IP addresses and simply rely on the load balancer for the service to be globally reachable. Many of these services do not require direct server return. This aids in address planning and utilization at scale since only the virtual IP (VIP) must be public.

## Overlay networks

The overlay functionality design includes OpenStack Networking in Open vSwitch GRE tunnel mode. In this case, the layer-3 external routers are paired with VRRP and switches should be paired with an implementation of MLAG running to ensure that you do not lose connectivity with the upstream routing infrastructure.

## Performance tuning

Network level tuning for this workload is minimal. Quality-of-Service (QoS) will be applied to these workloads for a middle ground Class Selector depending on existing policies. It will be higher than a best effort queue but lower than an Expedited Forwarding or Assured Forwarding queue. Since this type of application generates larger packets with longer-lived connections, bandwidth utilization can be optimized for long duration TCP. Normal bandwidth planning applies here with regards to benchmarking a session's usage multiplied by the expected number of concurrent sessions with overhead.

## Network functions

Network functions is a broad category but encompasses workloads that support the rest of a system's network. These workloads tend to consist

of large amounts of small packets that are very short lived, such as DNS queries or SNMP traps. These messages need to arrive quickly and do not deal with packet loss as there can be a very large volume of them. There are a few extra considerations to take into account for this type of workload and this can change a configuration all the way to the hypervisor level. For an application that generates 10 TCP sessions per user with an average bandwidth of 512 kilobytes per second per flow and expected user count of ten thousand concurrent users, the expected bandwidth plan is approximately 4.88 gigabits per second.

The supporting network for this type of configuration needs to have a low latency and evenly distributed availability. This workload benefits from having services local to the consumers of the service. A multi-site approach is used as well as deploying many copies of the application to handle load as close as possible to consumers. Since these applications function independently, they do not warrant running overlays to interconnect tenant networks. Overlays also have the drawback of performing poorly with rapid flow setup and may incur too much overhead with large quantities of small packets and are therefore not recommended.

QoS is desired for some workloads to ensure delivery. DNS has a major impact on the load times of other services and needs to be reliable and provide rapid responses. It is to configure rules in upstream devices to apply a higher Class Selector to DNS to ensure faster delivery or a better spot in queuing algorithms.

# Cloud storage

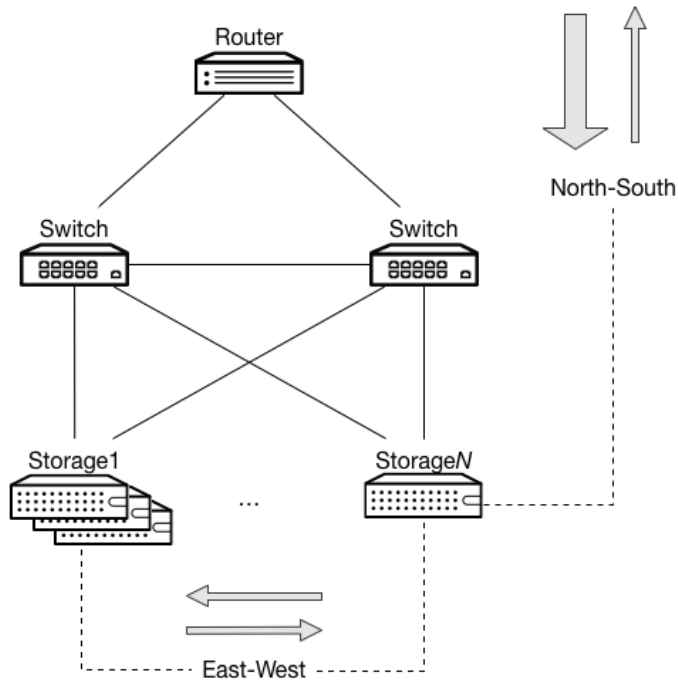
Another common use case for OpenStack environments is to provide a cloud-based file storage and sharing service. You might consider this a storage-focused use case, but its network-side requirements make it a network-focused use case.

For example, consider a cloud backup application. This workload has two specific behaviors that impact the network. Because this workload is an externally-facing service and an internally-replicating application, it has both *north-south* and *east-west traffic* considerations, as follows:

**north-south traffic** When a user uploads and stores content, that content moves into the OpenStack installation. When users download this content, the content moves from the OpenStack installation. Because this service is intended primarily as a backup, most of the traffic moves south-

bound into the environment. In this situation, it benefits you to configure a network to be asymmetrically downstream because the traffic that enters the OpenStack installation is greater than the traffic that leaves the installation.

**east-west traffic** Likely to be fully symmetric. Because replication originates from any node and might target multiple other nodes algorithmically, it is less likely for this traffic to have a larger volume in any specific direction. However this traffic might interfere with north-south traffic.




This application prioritizes the north-south traffic over east-west traffic: the north-south traffic involves customer-facing data.

The network design in this case is less dependant on availability and more dependant on being able to handle high bandwidth. As a direct result, it is beneficial to forego redundant links in favor of bonding those connections. This increases available bandwidth. It is also beneficial to configure

all devices in the path, including OpenStack, to generate and pass jumbo frames.

# 6. Multi-site

## Table of Contents



A multi-site OpenStack environment is one in which services located in more than one data center are used to provide the overall solution. Usage requirements of different multi-site clouds may vary widely, however they share some common needs. OpenStack is capable of running in a multi-region configuration allowing some parts of OpenStack to effectively manage a grouping of sites as a single cloud. With some careful planning in the design phase, OpenStack can act as an excellent multi-site cloud solution for a multitude of needs.

Some use cases that might indicate a need for a multi-site deployment of OpenStack include:

- An organization with a diverse geographic footprint.
- Geo-location sensitive data.
- Data locality, in which specific data or functionality should be close to users.

# User requirements

A multi-site architecture is complex and has its own risks and considerations, therefore it is important to make sure when contemplating the design such an architecture that it meets the user and business requirements.

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.

- Data ownership policies governing the possession and responsibility for data.

- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.

- Data compliance policies governing types of information that needs to reside in certain locations due to regular issues and, more importantly, cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union (http://ec.europa.eu/justice/data-protection) and the requirements of the Financial Industry Regulatory Authority (http://ec.europa.eu/justice/data-protection) in the United States. Consult a local regulatory body for more information.

# Workload characteristics

The expected workload is a critical requirement that needs to be captured to guide decision-making. An understanding of the workloads in the context of the desired multi-site environment and use case is important. Another way of thinking about a workload is to think of it as the way the systems are used. A workload could be a single application or a suite of applications that work together. It could also be a duplicate set of applications that need to run in multiple cloud environments. Often in a multi-site deployment the same workload will need to work identically in more than one physical location.

This multi-site scenario likely includes one or more of the other scenarios in this book with the additional requirement of having the workloads in two or more locations. The following are some possible scenarios:

For many use cases the proximity of the user to their workloads has a direct influence on the performance of the application and therefore should be taken into consideration in the design. Certain applications require zero to minimal latency that can only be achieved by deploying the cloud in multiple locations. These locations could be in different data centers, cities, countries or geographical regions, depending on the user requirement and location of the users.

# Consistency of images and templates across different sites

It is essential that the deployment of instances is consistent across the different sites. This needs to be built into the infrastructure. If OpenStack Object Store is used as a back end for the Image Service, it is possible to create repositories of consistent images across multiple sites. Having a central endpoint with multiple storage nodes will allow for a consistent centralized storage for each and every site.

Not using a centralized object store will increase operational overhead so that a consistent image library can be maintained. This could include development of a replication mechanism to handle the transport of images and the changes to the images across multiple sites.

# High availability

If high availability is a requirement to provide continuous infrastructure operations, a basic requirement of high availability should be defined.

The OpenStack management components need to have a basic and minimal level of redundancy. The simplest example is the loss of any single site has no significant impact on the availability of the OpenStack services of the entire infrastructure.

The *OpenStack High Availability Guide* contains more information on how to provide redundancy for the OpenStack components.

Multiple network links should be deployed between sites to provide redundancy for all components. This includes storage replication, which should be isolated to a dedicated network or VLAN with the ability to assign QoS to control the replication traffic or provide priority for this traffic. Note that if the data store is highly changeable, the network requirements could have a significant effect on the operational cost of maintaining the sites.

The ability to maintain object availability in both sites has significant implications on the object storage design and implementation. It will also have a significant impact on the WAN network design between the sites.

Connecting more than two sites increases the challenges and adds more complexity to the design considerations. Multi-site implementations require extra planning to address the additional topology complexity used

for internal and external connectivity. Some options include full mesh topology, hub spoke, spine leaf, or 3d Torus.

Not all the applications running in a cloud are cloud-aware. If that is the case, there should be clear measures and expectations to define what the infrastructure can support and, more importantly, what it cannot. An example would be shared storage between sites. It is possible, however such a solution is not native to OpenStack and requires a third-party hardware vendor to fulfill such a requirement. Another example can be seen in applications that are able to consume resources in object storage directly. These applications need to be cloud aware to make good use of an OpenStack Object Store.

## Application readiness

Some applications are tolerant of the lack of synchronized object storage, while others may need those objects to be replicated and available across regions. Understanding of how the cloud implementation impacts new and existing applications is important for risk mitigation and the overall success of a cloud project. Applications may have to be written to expect an infrastructure with little to no redundancy. Existing applications not developed with the cloud in mind may need to be rewritten.

## Cost

The requirement of having more than one site has a cost attached to it. The greater the number of sites, the greater the cost and complexity. Costs can be broken down into the following categories:

- Compute resources
- Networking resources
- Replication
- Storage
- Management
- Operational costs

## Site loss and recovery

Outages can cause loss of partial or full functionality of a site. Strategies should be implemented to understand and plan for recovery scenarios.

- The deployed applications need to continue to function and, more importantly, consideration should be taken of the impact on the performance and reliability of the application when a site is unavailable.

- It is important to understand what will happen to replication of objects and data between the sites when a site goes down. If this causes queues to start building up, considering how long these queues can safely exist until something explodes.

- Ensure determination of the method for resuming proper operations of a site when it comes back online after a disaster. It is recommended to architect the recovery to avoid race conditions.

## Compliance and geo-location

An organization could have certain legal obligations and regulatory compliance measures which could require certain workloads or data to not be located in certain regions.

## Auditing

A well thought-out auditing strategy is important in order to be able to quickly track down issues. Keeping track of changes made to security groups and tenant changes can be useful in rolling back the changes if they affect production. For example, if all security group rules for a tenant disappeared, the ability to quickly track down the issue would be important for operational and legal reasons.

## Separation of duties

A common requirement is to define different roles for the different cloud administration functions. An example would be a requirement to segregate the duties and permissions by site.

## Authentication between sites

Ideally it is best to have a single authentication domain and not need a separate implementation for each and every site. This will, of course, require an authentication mechanism that is highly available and distributed to ensure continuous operation. Authentication server locality is also something that might be needed as well and should be planned for.

# Technical considerations

There are many technical considerations to take into account with regard to designing a multi-site OpenStack implementation. An OpenStack cloud can be designed in a variety of ways to handle individual application needs. A multi-site deployment will have additional challenges compared to single site installations and will therefore be a more complex solution.

When determining capacity options be sure to take into account not just the technical issues, but also the economic or operational issues that might arise from specific decisions.

Inter-site link capacity describes the capabilities of the connectivity between the different OpenStack sites. This includes parameters such as bandwidth, latency, whether or not a link is dedicated, and any business policies applied to the connection. The capability and number of the links between sites will determine what kind of options may be available for deployment. For example, if two sites have a pair of high-bandwidth links available between them, it may be wise to configure a separate storage replication network between the two sites to support a single Swift endpoint and a shared object storage capability between them. (An example of this technique, as well as a configuration walk-through, is available at http://docs.openstack.org/developer/swift/replication_network.html#dedicated-replication-network). Another option in this scenario is to build a dedicated set of tenant private networks across the secondary link using overlay networks with a third party mapping the site overlays to each other.

The capacity requirements of the links between sites will be driven by application behavior. If the latency of the links is too high, certain applications that use a large number of small packets, for example RPC calls, may encounter issues communicating with each other or operating properly. Additionally, OpenStack may encounter similar types of issues. To mitigate this, tuning of the Identity service call timeouts may be necessary to prevent issues authenticating against a central Identity service.

Another capacity consideration when it comes to networking for a multi-site deployment is the available amount and performance of overlay networks for tenant networks. If using shared tenant networks across zones, it is imperative that an external overlay manager or controller be used to map these overlays together. It is necessary to ensure the amount of possible IDs between the zones are identical. Note that, as of the Icehouse release, OpenStack Networking was not capable of managing tunnel IDs

across installations. This means that if one site runs out of IDs, but other does not, that tenant's network will be unable to reach the other site.

Capacity can take other forms as well. The ability for a region to grow depends on scaling out the number of available compute nodes. This topic is covered in greater detail in the section for compute-focused deployments. However, it should be noted that cells may be necessary to grow an individual region beyond a certain point. This point depends on the size of your cluster and the ratio of virtual machines per hypervisor.

A third form of capacity comes in the multi-region-capable components of OpenStack. Centralized Object Storage is capable of serving objects through a single namespace across multiple regions. Since this works by accessing the object store via swift proxy, it is possible to overload the proxies. There are two options available to mitigate this issue. The first is to deploy a large number of swift proxies. The drawback to this is that the proxies are not load-balanced and a large file request could continually hit the same proxy. The other way to mitigate this is to front-end the proxies with a caching HTTP proxy and load balancer. Since swift objects are returned to the requester via HTTP, this load balancer would alleviate the load required on the swift proxies.

# Utilization

While constructing a multi-site OpenStack environment is the goal of this guide, the real test is whether an application can utilize it.

Identity is normally the first interface for the majority of OpenStack users. Interacting with the Identity service is required for almost all major operations within OpenStack. Therefore, it is important to ensure that you provide users with a single URL for Identity service authentication. Equally important is proper documentation and configuration of regions within the Identity service. Each of the sites defined in your installation is considered to be a region in Identity nomenclature. This is important for the users of the system, when reading Identity documentation, as it is required to define the region name when providing actions to an API endpoint or in the dashboard.

Load balancing is another common issue with multi-site installations. While it is still possible to run HAproxy instances with Load-Balancer-as-a-Service, these will be local to a specific region. Some applications may be able to cope with this via internal mechanisms. Others, however, may require the implementation of an external system including global services load balancers or anycast-advertised DNS.

Depending on the storage model chosen during site design, storage replication and availability will also be a concern for end-users. If an application is capable of understanding regions, then it is possible to keep the object storage system separated by region. In this case, users who want to have an object available to more than one region will need to do the cross-site replication themselves. With a centralized swift proxy, however, the user may need to benchmark the replication timing of the Object Storage back end. Benchmarking allows the operational staff to provide users with an understanding of the amount of time required for a stored or modified object to become available to the entire environment.

# Performance

Determining the performance of a multi-site installation involves considerations that do not come into play in a single-site deployment. Being a distributed deployment, multi-site deployments incur a few extra penalties to performance in certain situations.

Since multi-site systems can be geographically separated, they may have worse than normal latency or jitter when communicating across regions. This can especially impact systems like the OpenStack Identity service when making authentication attempts from regions that do not contain the centralized Identity implementation. It can also affect certain applications which rely on remote procedure call (RPC) for normal operation. An example of this can be seen in High Performance Computing workloads.

Storage availability can also be impacted by the architecture of a multi-site deployment. A centralized Object Storage service requires more time for an object to be available to instances locally in regions where the object was not created. Some applications may need to be tuned to account for this effect. Block Storage does not currently have a method for replicating data across multiple regions, so applications that depend on available block storage will need to manually cope with this limitation by creating duplicate block storage entries in each region.

# Security

Securing a multi-site OpenStack installation also brings extra challenges. Tenants may expect a tenant-created network to be secure. In a multi-site installation the use of a non-private connection between sites may be required. This may mean that traffic would be visible to third parties and, in cases where an application requires security, this issue will require mitigation. Installing a VPN or encrypted connection between sites is recommended in such instances.

Another security consideration with regard to multi-site deployments is Identity. Authentication in a multi-site deployment should be centralized. Centralization provides a single authentication point for users across the deployment, as well as a single point of administration for traditional create, read, update and delete operations. Centralized authentication is also useful for auditing purposes because all authentication tokens originate from the same source.

Just as tenants in a single-site deployment need isolation from each other, so do tenants in multi-site installations. The extra challenges in multi-site designs revolve around ensuring that tenant networks function across regions. Unfortunately, OpenStack Networking does not presently support a mechanism to provide this functionality, therefore an external system may be necessary to manage these mappings. Tenant networks may contain sensitive information requiring that this mapping be accurate and consistent to ensure that a tenant in one site does not connect to a different tenant in another site.

# OpenStack components

Most OpenStack installations require a bare minimum set of pieces to function. These include the OpenStack Identity (keystone) for authentication, OpenStack Compute (nova) for compute, OpenStack Image Service (glance) for image storage, OpenStack Networking (neutron) for networking, and potentially an object store in the form of OpenStack Object Storage (swift). Bringing multi-site into play also demands extra components in order to coordinate between regions. Centralized Identity service is necessary to provide the single authentication point. Centralized dashboard is also recommended to provide a single login point and a mapped experience to the API and CLI options available. If necessary, a centralized Object Storage service may be used and will require the installation of the swift proxy service.

It may also be helpful to install a few extra options in order to facilitate certain use cases. For instance, installing designate may assist in automatically generating DNS domains for each region with an automatically-populated zone full of resource records for each instance. This facilitates using DNS as a mechanism for determining which region would be selected for certain applications.

Another useful tool for managing a multi-site installation is Orchestration (heat). The Orchestration module allows the use of templates to define a set of instances to be launched together or for scaling existing sets. It can also be used to setup matching or differentiated groupings based on re-

gions. For instance, if an application requires an equally balanced number of nodes across sites, the same heat template can be used to cover each site with small alterations to only the region name.

# Operational considerations

Deployment of a multi-site OpenStack cloud using regions requires that the service catalog contains per-region entries for each service deployed other than the Identity service itself. There is limited support amongst currently available off-the-shelf OpenStack deployment tools for defining multiple regions in this fashion.

Deployers must be aware of this and provide the appropriate customization of the service catalog for their site either manually or via customization of the deployment tools in use.

Note that, as of the Icehouse release, documentation for implementing this feature is in progress. See this bug for more information: https://bugs.launchpad.net/openstack-manuals/+bug/1340509.

# Licensing

Multi-site OpenStack deployments present additional licensing considerations over and above regular OpenStack clouds, particularly where site licenses are in use to provide cost efficient access to software licenses. The licensing for host operating systems, guest operating systems, OpenStack distributions (if applicable), software-defined infrastructure including network controllers and storage systems, and even individual applications need to be evaluated in light of the multi-site nature of the cloud.

Topics to consider include:

- The specific definition of what constitutes a site in the relevant licenses, as the term does not necessarily denote a geographic or otherwise physically isolated location in the traditional sense.
- Differentiations between "hot" (active) and "cold" (inactive) sites where significant savings may be made in situations where one site is a cold standby for disaster recovery purposes only.
- Certain locations might require local vendors to provide support and services for each site provides challenges, but will vary on the licensing agreement in place.

# Logging and monitoring

Logging and monitoring does not significantly differ for a multi-site OpenStack cloud. The same well known tools described in the Logging and monitoring chapter of the *Operations Guide* remain applicable. Logging and monitoring can be provided both on a per-site basis and in a common centralized location.

When attempting to deploy logging and monitoring facilities to a centralized location, care must be taken with regards to the load placed on the inter-site networking links.

# Upgrades

In multi-site OpenStack clouds deployed using regions each site is, effectively, an independent OpenStack installation which is linked to the others by using centralized services such as Identity which are shared between sites. At a high level the recommended order of operations to upgrade an individual OpenStack environment is (see the Upgrades chapter of the *Operations Guide* for details):

1. Upgrade the OpenStack Identity service (keystone).
2. Upgrade the OpenStack Image Service (glance).
3. Upgrade OpenStack Compute (nova), including networking components.
4. Upgrade OpenStack Block Storage (cinder).
5. Upgrade the OpenStack dashboard (horizon).

The process for upgrading a multi-site environment is not significantly different:

1. Upgrade the shared OpenStack Identity service (keystone) deployment.
2. Upgrade the OpenStack Image Service (glance) at each site.
3. Upgrade OpenStack Compute (nova), including networking components, at each site.
4. Upgrade OpenStack Block Storage (cinder) at each site.

5. Upgrade the OpenStack dashboard (horizon), at each site or in the single central location if it is shared.

Note that, as of the OpenStack Icehouse release, compute upgrades within each site can also be performed in a rolling fashion. Compute controller services (API, Scheduler, and Conductor) can be upgraded prior to upgrading of individual compute nodes. This maximizes the ability of operations staff to keep a site operational for users of compute services while performing an upgrade.

# Quota management

To prevent system capacities from being exhausted without notification, OpenStack provides operators with the ability to define quotas. Quotas are used to set operational limits and are currently enforced at the tenant (or project) level rather than at the user level.

Quotas are defined on a per-region basis. Operators may wish to define identical quotas for tenants in each region of the cloud to provide a consistent experience, or even create a process for synchronizing allocated quotas across regions. It is important to note that only the operational limits imposed by the quotas will be aligned consumption of quotas by users will not be reflected between regions.

For example, given a cloud with two regions, if the operator grants a user a quota of 25 instances in each region then that user may launch a total of 50 instances spread across both regions. They may not, however, launch more than 25 instances in any single region.

For more information on managing quotas refer to the Managing projects and users chapter of the *OpenStack Operators Guide*.

# Policy management

OpenStack provides a default set of Role Based Access Control (RBAC) policies, defined in a `policy.json` file, for each service. Operators edit these files to customize the policies for their OpenStack installation. If the application of consistent RBAC policies across sites is considered a requirement, then it is necessary to ensure proper synchronization of the `policy.json` files to all installations.

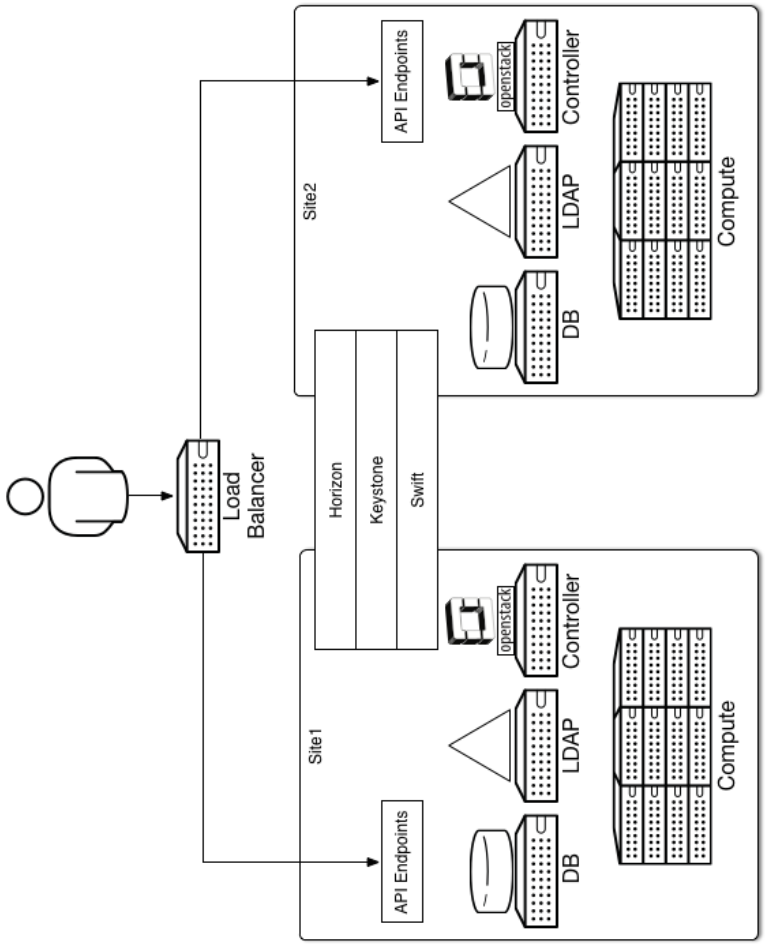
This must be done using normal system administration tools such as rsync as no functionality for synchronizing policies across regions is currently provided within OpenStack.

# Documentation

Users must be able to leverage cloud infrastructure and provision new resources in the environment. It is important that user documentation is accessible by users of the cloud infrastructure to ensure they are given sufficient information to help them leverage the cloud. As an example, by default OpenStack will schedule instances on a compute node automatically. However, when multiple regions are available, it is left to the end user to decide in which region to schedule the new instance. The dashboard will present the user with the first region in your configuration. The API and CLI tools will not execute commands unless a valid region is specified. It is therefore important to provide documentation to your users describing the region layout as well as calling out that quotas are region-specific. If a user reaches his or her quota in one region, OpenStack will not automatically build new instances in another. Documenting specific examples will help users understand how to operate the cloud, thereby reducing calls and tickets filed with the help desk.

# Architecture

This graphic is a high level diagram of a multiple site OpenStack architecture. Each site is an OpenStack cloud but it may be necessary to architect the sites on different versions. For example, if the second site is intended to be a replacement for the first site, they would be different. Another common design would be a private OpenStack cloud with replicated site that would be used for high availability or disaster recovery. The most important design decision is how to configure the storage. It can be configured as a single shared pool or separate pools, depending on the user and technical requirements.




# OpenStack services architecture

The OpenStack Identity service, which is used by all other OpenStack components for authorization and the catalog of service endpoints, supports the concept of regions. A region is a logical construct that can be used to group OpenStack services that are in close proximity to one another. The concept of regions is flexible; it may can contain OpenStack service endpoints located within a distinct geographic region, or regions. It may be smaller in scope, where a region is a single rack within a data center or even a single blade chassis, with multiple regions existing in adjacent racks in the same data center.

The majority of OpenStack components are designed to run within the context of a single region. The OpenStack Compute service is designed to manage compute resources within a region, with support for subdivisions of compute resources by using availability zones and cells. The OpenStack Networking service can be used to manage network resources in the same broadcast domain or collection of switches that are linked. The OpenStack Block Storage service controls storage resources within a region with all storage resources residing on the same storage network. Like the OpenStack Compute service, the OpenStack Block Storage service also supports the availability zone construct which can be used to subdivide storage resources.

The OpenStack dashboard, OpenStack Identity, and OpenStack Object Storage services are components that can each be deployed centrally in order to serve multiple regions.

# Storage

With multiple OpenStack regions, having a single OpenStack Object Storage service endpoint that delivers shared file storage for all regions is desirable. The Object Storage service internally replicates files to multiple nodes. The advantages of this are that, if a file placed into the Object Storage service is visible to all regions, it can be used by applications or workloads in any or all of the regions. This simplifies high availability failover and disaster recovery rollback.

In order to scale the Object Storage service to meet the workload of multiple regions, multiple proxy workers are run and load-balanced, storage nodes are installed in each region, and the entire Object Storage Service can be fronted by an HTTP caching layer. This is done so client requests for objects can be served out of caches rather than directly from the storage modules themselves, reducing the actual load on the storage network. In addition to an HTTP caching layer, use a caching layer like Memcache to cache objects between the proxy and storage nodes.

If the cloud is designed without a single Object Storage Service endpoint for multiple regions, and instead a separate Object Storage Service endpoint is made available in each region, applications are required to handle synchronization (if desired) and other management operations to ensure consistency across the nodes. For some applications, having multiple Object Storage Service endpoints located in the same region as the application may be desirable due to reduced latency, cross region bandwidth, and ease of deployment.

For the Block Storage service, the most important decisions are the selection of the storage technology and whether or not a dedicated network is used to carry storage traffic from the storage service to the compute nodes.

## Networking

When connecting multiple regions together there are several design considerations. The overlay network technology choice determines how packets are transmitted between regions and how the logical network and addresses present to the application. If there are security or regulatory requirements, encryption should be implemented to secure the traffic between regions. For networking inside a region, the overlay network technology for tenant networks is equally important. The overlay technology and the network traffic of an application generates or receives can be either complementary or be at cross purpose. For example, using an overlay technology for an application that transmits a large amount of small packets could add excessive latency or overhead to each packet if not configured properly.

## Dependencies

The architecture for a multi-site installation of OpenStack is dependent on a number of factors. One major dependency to consider is storage. When designing the storage system, the storage mechanism needs to be determined. Once the storage type is determined, how it will be accessed is critical. For example, it is recommended that storage should utilize a dedicated network. Another concern is how the storage is configured to protect the data. For example, the recovery point objective (RPO) and the recovery time objective (RTO). How quickly can the recovery from a fault be completed, will determine how often the replication of data be required. Ensure that enough storage is allocated to support the data protection strategy.

Networking decisions include the encapsulation mechanism that will be used for the tenant networks, how large the broadcast domains should be, and the contracted SLAs for the interconnects.

# Prescriptive examples

Based on the needs of the intended workloads, there are multiple ways to build a multi-site OpenStack installation. Below are example architec-

tures based on different requirements. These examples are meant as a reference, and not a hard and fast rule for deployments. Use the previous sections of this chapter to assist in selecting specific components and implementations based on specific needs.

A large content provider needs to deliver content to customers that are geographically dispersed. The workload is very sensitive to latency and needs a rapid response to end-users. After reviewing the user, technical and operational considerations, it is determined beneficial to build a number of regions local to the customer's edge. In this case rather than build a few large, centralized data centers, the intent of the architecture is to provide a pair of small data centers in locations that are closer to the customer. In this use case, spreading applications out allows for different horizontal scaling than a traditional compute workload scale. The intent is to scale by creating more copies of the application in closer proximity to the users that need it most, in order to ensure faster response time to user requests. This provider will deploy two datacenters at each of the four chosen regions. The implications of this design are based around the method of placing copies of resources in each of the remote regions. Swift objects, Glance images, and block storage will need to be manually replicated into each region. This may be beneficial for some systems, such as the case of content service, where only some of the content needs to exist in some but not all regions. A centralized Keystone is recommended to ensure authentication and that access to the API endpoints is easily manageable.

Installation of an automated DNS system such as Designate is highly recommended. Unless an external Dynamic DNS system is available, application administrators will need a way to manage the mapping of which application copy exists in each region and how to reach it. Designate will assist by making the process automatic and by populating the records in the each region's zone.

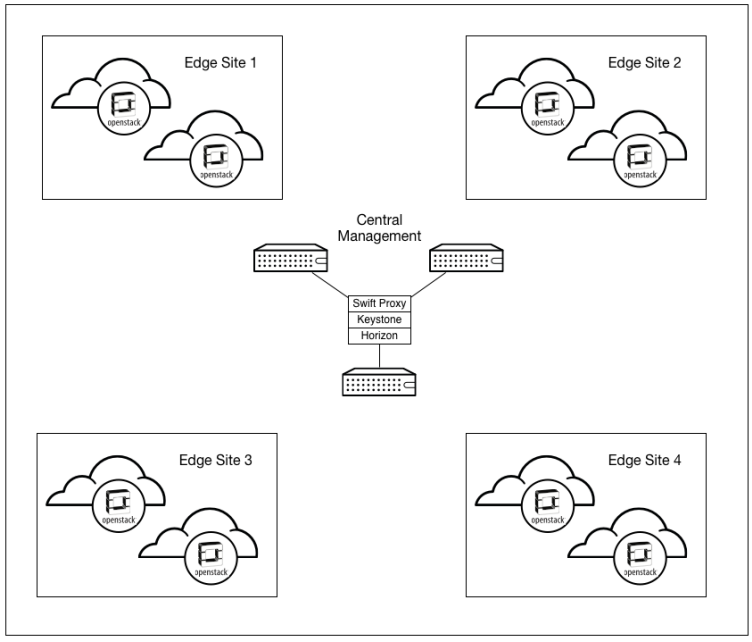
Telemetry for each region is also deployed, as each region may grow differently or be used at a different rate. Ceilometer will run to collect each region's metrics from each of the controllers and report them back to a central location. This is useful both to the end user and the administrator of the OpenStack environment. The end user will find this method useful, in that it is possible to determine if certain locations are experiencing higher load than others, and take appropriate action. Administrators will also benefit by possibly being able to forecast growth per region, rather than expanding the capacity of all regions simultaneously, therefore maximizing the cost-effectiveness of the multi-site design.

One of the key decisions of running this sort of infrastructure is whether or not to provide a redundancy model. Two types of redundancy and high

availability models in this configuration will be implemented. The first type revolves around the availability of the central OpenStack components. Keystone will be made highly available in three central data centers that will host the centralized OpenStack components. This prevents a loss of any one of the regions causing an outage in service. It also has the added benefit of being able to run a central storage repository as a primary cache for distributing content to each of the regions.

The second redundancy topic is that of the edge data center itself. A second data center in each of the edge regional locations will house a second region near the first. This ensures that the application will not suffer degraded performance in terms of latency and availability.

This figure depicts the solution designed to have both a centralized set of core data centers for OpenStack services and paired edge data centers:




# Geo-redundant load balancing

A large-scale web application has been designed with cloud principles in mind. The application is designed provide service to application store, on a 24/7 basis. The company has typical 2-tier architecture with a web front-end servicing the customer requests and a NoSQL database back end storing the information.

As of late there has been several outages in number of major public cloud providers—usually due to the fact these applications were running out of a single geographical location. The design therefore should mitigate the chance of a single site causing an outage for their business.

The solution would consist of the following OpenStack components:

- A firewall, switches and load balancers on the public facing network connections.
- OpenStack Controller services running, Networking, dashboard, Block Storage and Compute running locally in each of the three regions. The other services, Identity, Orchestration, Telemetry, Image Service and Object Storage will be installed centrally—with nodes in each of the region providing a redundant OpenStack Controller plane throughout the globe.
- OpenStack Compute nodes running the KVM hypervisor.
- OpenStack Object Storage for serving static objects such as images will be used to ensure that all images are standardized across all the regions, and replicated on a regular basis.
- A Distributed DNS service available to all regions—that allows for dynamic update of DNS records of deployed instances.
- A geo-redundant load balancing service will be used to service the requests from the customers based on their origin.

An autoscaling heat template will used to deploy the application in the three regions. This template will include:

- Web Servers, running Apache.
- Appropriate `user_data` to populate the central DNS servers upon instance launch.
- Appropriate Telemetry alarms that maintain state of the application and allow for handling of region or instance failure.

Another autoscaling Heat template will be used to deploy a distributed MongoDB shard over the three locations—with the option of storing required data on a globally available swift container. according to the usage and load on the database server—additional shards will be provisioned according to the thresholds defined in Telemetry.

The reason that three regions were selected here was because of the fear of having abnormal load on a single region in the event of a failure. Two data center would have been sufficient had the requirements been met.

Orchestration is used because of the built-in functionality of autoscaling and auto healing in the event of increased load. Additional configuration management tools, such as Puppet or Chef could also have been used in this scenario, but were not chosen due to the fact that Orchestration had the appropriate built-in hooks into the OpenStack cloud—whereas the other tools were external and not native to OpenStack. In addition—since this deployment scenario was relatively straight forward—the external tools were not needed.

OpenStack Object Storage is used here to serve as a back end for the Image Service since was the most suitable solution for a globally distributed storage solution—with its own replication mechanism. Home grown solutions could also have been used including the handling of replication—but were not chosen, because Object Storage is already an intricate part of the infrastructure—and proven solution.

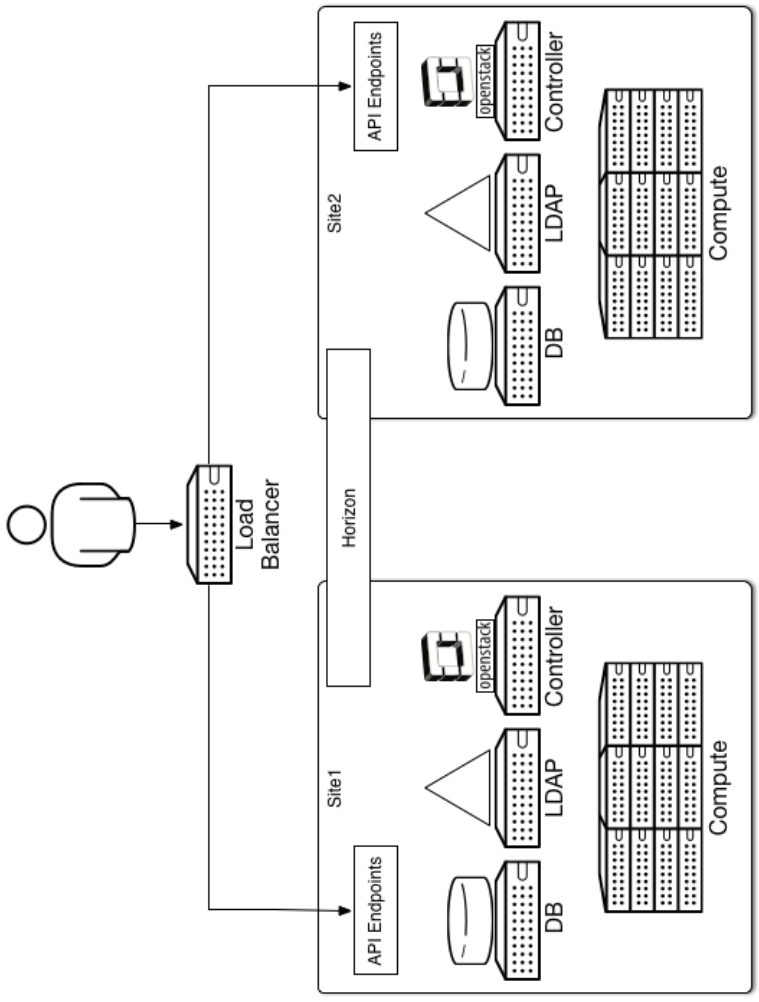
An external load balancing service was used and not the LBaaS in OpenStack because the solution in OpenStack is not redundant and does not have any awareness of geo location.

# Location-local service

A common use for a multi-site deployment of OpenStack, is for creating a Content Delivery Network. An application that uses a location-local architecture will require low network latency and proximity to the user, in order to provide an optimal user experience, in addition to reducing the cost of bandwidth and transit, since the content resides on sites closer to the customer, instead of a centralized content store that would require utilizing higher cost cross country links.

This architecture usually includes a geo-location component that places user requests at the closest possible node. In this scenario, 100% redundancy of content across every site is a goal rather than a requirement, with the intent being to maximize the amount of content available that is within a minimum number of network hops for any given end user. Despite these differences, the storage replication configuration has significant overlap with that of a geo-redundant load balancing use case.

In this example, the application utilizing this multi-site OpenStack install that is location aware would launch web server or content serving instances on the compute cluster in each site. Requests from clients will first be sent to a global services load balancer that determines the location of the client, then routes the request to the closest OpenStack site where the application completes the request.


Load Balancer
Horizon
Site1
API Endpoints
DB
LDAP
openstack
Controller
Compute
Site2
API Endpoints
DB
LDAP
openstack
Controller
Compute

# 7. Hybrid

## Table of Contents



*Hybrid cloud*, by definition, means that the design spans more than one cloud. An example of this kind of architecture may include a situation in which the design involves more than one OpenStack cloud (for example, an OpenStack-based private cloud and an OpenStack-based public cloud), or it may be a situation incorporating an OpenStack cloud and a non-OpenStack cloud (for example, an OpenStack-based private cloud that interacts with Amazon Web Services). *Bursting* into an external cloud is the practice of creating new instances to alleviate extra load where there is no available capacity in the private cloud.

Some situations that could involve hybrid cloud architecture include:

- Bursting from a private cloud to a public cloud
- Disaster recovery
- Development and testing
- Federated cloud, enabling users to choose resources from multiple providers
- Hybrid clouds built to support legacy systems as they transition to cloud

As a hybrid cloud design deals with systems that are outside of the control of the cloud architect or organization, a hybrid cloud architecture requires considering aspects of the architecture that might not have otherwise been necessary. For example, the design may need to deal with hardware, software, and APIs under the control of a separate organization.

Similarly, the degree to which the architecture is OpenStack-based will have an effect on the cloud operator or cloud consumer's ability to accomplish tasks with native OpenStack tools. By definition, this is a situation in which no single cloud can provide all of the necessary functionality. In

order to manage the entire system, users, operators and consumers will need an overarching tool known as a cloud management platform (CMP). Any organization that is working with multiple clouds already has a CMP, even if that CMP is the operator who logs into an external web portal and launches a public cloud instance.

There are commercially available options, such as Rightscale, and open source options, such as ManageIQ (http://manageiq.org), but there is no single CMP that can address all needs in all scenarios. Whereas most of the sections of this book talk about the aspects of OpenStack, an architect needs to consider when designing an OpenStack architecture. This section will also discuss the things the architect must address when choosing or building a CMP to run a hybrid cloud design, even if the CMP will be a manually built solution.

# User requirements

Hybrid cloud architectures introduce additional complexities, particularly those that use heterogeneous cloud platforms. As a result, it is important to make sure that design choices match requirements in such a way that the benefits outweigh the inherent additional complexity and risks.

Business considerations to make when designing a hybrid cloud deployment include:

**Cost**

A hybrid cloud architecture involves multiple vendors and technical architectures. These architectures may be more expensive to deploy and maintain. Operational costs can be higher because of the need for more sophisticated orchestration and brokerage tools than in other architectures. In contrast, overall operational costs might be lower by virtue of using a cloud brokerage tool to deploy the workloads to the most cost effective platform.

**Revenue opportunity**

Revenue opportunities vary greatly based on the intent and use case of the cloud. If it is being built as a commercial customer-facing product, consider the drivers for building it over multiple platforms and whether the use of mul-

| | |
|---|---|
| | tiple platforms make the design more attractive to target customers, thus enhancing the revenue opportunity. |
| **Time to market** | One of the most common reasons to use cloud platforms is to speed the time to market of a new product or application. A business requirement to use multiple cloud platforms may be because there is an existing investment in several applications and it is faster to tie them together rather than migrating components and refactoring to a single platform. |
| **Business or technical diversity** | Organizations already leveraging cloud-based services may wish to embrace business diversity and utilize a hybrid cloud design to spread their workloads across multiple cloud providers so that no application is hosted in a single cloud provider. |
| **Application momentum** | A business with existing applications that are already in production on multiple cloud environments may find that it is more cost effective to integrate the applications on multiple cloud platforms rather than migrate them to a single platform. |

# Legal requirements

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.
- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.

- Data compliance policies governing certain types of information needs to reside in certain locations due to regular issues and, more importantly, cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union (http://ec.europa.eu/justice/data-protection/) and the requirements of the Financial Industry Regulatory Authority (http://www.finra.org/Industry/Regulation/FINRARules/) in the United States. Consult a local regulatory body for more information.

# Workload considerations

Defining what the word "workload" means in the context of a hybrid cloud environment is important. Workload can be defined as the intended way the systems will be utilized, which is often referred to as a "use case". A workload can be a single application or a suite of applications that work in concert. It can also be a duplicate set of applications that need to run on multiple cloud environments. In a hybrid cloud deployment, the same workload will often need to function equally well on radically different public and private cloud environments. The architecture needs to address these potential conflicts, complexity, and platform incompatibilities.

Some possible use cases for a hybrid cloud architecture include:

- Dynamic resource expansion or "bursting": Another common reason to use a multiple cloud architecture is a "bursty" application that needs additional resources at times. An example of this case could be a retailer that needs additional resources during the holiday selling season, but does not want to build expensive cloud resources to meet the peak demand. They might have an OpenStack private cloud but want to burst to AWS or some other public cloud for these peak load periods. These bursts could be for long or short cycles ranging from hourly, monthly or yearly.

- Disaster recovery-business continuity: The cheaper storage and instance management makes a good case for using the cloud as a secondary site. The public cloud is already heavily used for these purposes in combination with an OpenStack public or private cloud.

- Federated hypervisor-instance management: Adding self-service, charge back and transparent delivery of the right resources from a federated pool can be cost effective. In a hybrid cloud environment, this is a particularly important consideration. Look for a cloud that provides cross-platform hypervisor support and robust instance management tools.

- Application portfolio integration: An enterprise cloud delivers better application portfolio management and more efficient deployment by leveraging self-service features and rules for deployments based on types of use. A common driver for building hybrid cloud architecture is to stitch together multiple existing cloud environments that are already in production or development.

- Migration scenarios: A common reason to create a hybrid cloud architecture is to allow the migration of applications between different clouds. This may be because the application will be migrated permanently to a new platform, or it might be because the application needs to be supported on multiple platforms going forward.

- High availability: Another important reason for wanting a multiple cloud architecture is to address the needs for high availability. By using a combination of multiple locations and platforms, a design can achieve a level of availability that is not possible with a single platform. This approach does add a significant amount of complexity.

In addition to thinking about how the workload will work on a single cloud, the design must accommodate the added complexity of needing the workload to run on multiple cloud platforms. The complexity of transferring workloads across clouds needs to be explored at the application, instance, cloud platform, hypervisor, and network levels.

# Tools considerations

When working with designs spanning multiple clouds, the design must incorporate tools to facilitate working across those multiple clouds. Some of the user requirements drive the need for tools that will do the following functions:

- Broker between clouds: Since the multiple cloud architecture assumes that there will be at least two different and possibly incompatible platforms that are likely to have different costs, brokering software is designed to evaluate relative costs between different cloud platforms. These solutions are sometimes referred to as Cloud Management Platforms (CMPs). Examples include Rightscale, Gravitent, Scalr, CloudForms, and ManageIQ. These tools allow the designer to determine the right location for the workload based on predetermined criteria.

- Facilitate orchestration across the clouds: CMPs are tools are used to tie everything together. Cloud orchestration tools are used to improve the management of IT application portfolios as they migrate onto public,

private, and hybrid cloud platforms. These tools are an important consideration. Cloud orchestration tools are used for managing a diverse portfolio of installed systems across multiple cloud platforms. The typical enterprise IT application portfolio is still comprised of a few thousand applications scattered over legacy hardware, virtualized infrastructure, and now dozens of disjointed shadow public Infrastructure-as-a-Service (IaaS) and Software-as-a-Service (SaaS) providers and offerings.

# Network considerations

The network services functionality is an important factor to assess when choosing a CMP and cloud provider. Considerations are functionality, security, scalability and HA. Verification and ongoing testing of the critical features of the cloud endpoint used by the architecture are important tasks.

- Once the network functionality framework has been decided, a minimum functionality test should be designed. This will ensure testing and functionality persists during and after upgrades.

- Scalability across multiple cloud providers may dictate which underlying network framework you will choose in different cloud providers. It is important to have the network API functions presented and to verify that functionality persists across all cloud endpoints chosen.

- High availability implementations vary in functionality and design. Examples of some common methods are active-hot-standby, active-passive and active-active. High availability and a test framework needs to be developed to insure that the functionality and limitations are well understood.

- Security considerations include how data is secured between client and endpoint and any traffic that traverses the multiple clouds, from eavesdropping to DoS activities.

# Risk mitigation and management considerations

Hybrid cloud architectures introduce additional risk because they add additional complexity and potentially conflicting or incompatible components or tools. However, they also reduce risk by spreading workloads over multiple providers. This means, if one was to go out of business, the organization could remain operational.

Risks that will be heightened by using a hybrid cloud architecture include:

| | |
|---|---|
| **Provider availability or implementation details** | This can range from the company going out of business to the company changing how it delivers its services. Cloud architectures are inherently designed to be flexible and changeable; paradoxically, the cloud is both perceived to be rock solid and ever flexible at the same time. |
| **Differing SLAs** | Users of hybrid cloud environments potentially encounter some losses through differences in service level agreements. A hybrid cloud design needs to accommodate the different SLAs provided by the various clouds involved in the design, and must address the actual enforceability of the providers' SLAs. |
| **Security levels** | Securing multiple cloud environments is more complex than securing a single cloud environment. Concerns need to be addressed at, but not limited to, the application, network, and cloud platform levels. One issue is that different cloud platforms approach security differently, and a hybrid cloud design must address and compensate for differences in security approaches. For example, AWS uses a relatively simple model that relies on user privilege combined with firewalls. |
| **Provider API changes** | APIs are crucial in a hybrid cloud environment. As a consumer of a provider's cloud services, an organization will rarely have any control over provider changes to APIs. Cloud services that might have previously had compatible APIs may no longer work. This is particularly a problem with AWS and OpenStack AWS-compatible APIs. OpenStack was originally planned to maintain compatibility with changes in AWS |

APIs. However, over time, the APIs have become more divergent in functionality. One way to address this issue is to focus on using only the most common and basic APIs to minimize potential conflicts.

# Technical considerations

A hybrid cloud environment requires inspection and understanding of technical issues that are not only outside of an organization's data center, but potentially outside of an organization's control. In many cases, it is necessary to ensure that the architecture and CMP chosen can adapt to, not only to different environments, but also to the possibility of change. In this situation, applications are crossing diverse platforms and are likely to be located in diverse locations. All of these factors will influence and add complexity to the design of a hybrid cloud architecture.

The only situation where cloud platform incompatibilities are not going to be an issue is when working with clouds that are based on the same version and the same distribution of OpenStack. Otherwise incompatibilities are virtually inevitable.

Incompatibility should be less of an issue for clouds that exclusively use the same version of OpenStack, even if they use different distributions. The newer the distribution in question, the less likely it is that there will be incompatibilities between version. This is due to the fact that the OpenStack community has established an initiative to define core functions that need to remain backward compatible between supported versions. The DefCore initiative defines basic functions that every distribution must support in order to bear the name "OpenStack".

Some vendors, however, add proprietary customizations to their distributions. If an application or architecture makes use of these features, it will be difficult to migrate to or use other types of environments. Anyone considering incorporating older versions of OpenStack prior to Havana should consider carefully before attempting to incorporate functionality between versions. Internal differences in older versions may be so great that the best approach might be to consider the versions to be essentially diverse platforms, as different as OpenStack and Amazon Web Services or Microsoft Azure.

The situation is more predictable if using different cloud platforms is incorporated from inception. If the other clouds are not based on OpenStack,

then all pretense of compatibility vanishes, and CMP tools must account for the myriad of differences in the way operations are handled and services are implemented. Some situations in which these incompatibilities can arise include differences between the way in which a cloud:

- Deploys instances
- Manages networks
- Treats applications
- Implements services

# Capacity planning

One of the primary reasons many organizations turn to a hybrid cloud system is to increase capacity without having to make large capital investments. However, capacity planning is still necessary when designing an OpenStack installation even if it is augmented with external clouds.

Specifically, overall capacity and placement of workloads need to be accounted for when designing for a mostly internally-operated cloud with the occasional capacity burs. The long-term capacity plan for such a design needs to incorporate growth over time to prevent the need to permanently burst into, and occupy, a potentially more expensive external cloud. In order to avoid this scenario, account for the future applications and capacity requirements and plan growth appropriately.

One of the drawbacks of capacity planning is unpredictability. It is difficult to predict the amount of load a particular application might incur if the number of users fluctuates or the application experiences an unexpected increase in popularity. It is possible to define application requirements in terms of vCPU, RAM, bandwidth or other resources and plan appropriately, but other clouds may not use the same metric or even the same oversubscription rates.

Oversubscription is a method to emulate more capacity than they may physically be present. For example, a physical hypervisor node with 32 GB RAM may host 24 instances, each provisioned with 2 GB RAM. As long as all 24 of them are not concurrently utilizing 2 full gigabytes, this arrangement is a non-issue. However, some hosts take oversubscription to extremes and, as a result, performance can frequently be inconsistent. If at all possible, determine what the oversubscription rates of each host are and plan capacity accordingly.

# Security

The nature of a hybrid cloud environment removes complete control over the infrastructure. Security becomes a stronger requirement because data or applications may exist in a cloud that is outside of an organization's control. Security domains become an important distinction when planning for a hybrid cloud environment and its capabilities. A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system.

The security domains are:

1. Public
2. Guest
3. Management
4. Data

These security domains can be mapped individually to the organization's installation or combined. For example, some deployment topologies combine both guest and data domains onto one physical network, whereas other topologies may physically separate these networks. In each case, the cloud operator should be aware of the appropriate security concerns. Security domains should be mapped out against the specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which an organization has no authority. This domain should always be considered untrusted. When considering hybrid cloud deployments, any traffic traversing beyond and between the multiple clouds should always be considered to reside in this security domain and is therefore untrusted.

Typically used for instance-to-instance traffic within a single data center, the guest security domain handles compute data generated by instances on the cloud but not services that support the operation of the cloud such as API calls. Public cloud providers that are used in a hybrid cloud configuration which an organization does not control and private cloud providers who do not have stringent controls on instance use or who allow unrestricted Internet access to instances should consider this domain to be untrusted. Private cloud providers may consider this network as internal and therefore trusted only if there are controls in place to assert that instances and tenants are trusted.

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, user names, and passwords. In deployments behind an organization's firewall, this domain is considered trusted. In a public cloud model which could be part of an architecture, this would have to be assessed with the public cloud provider to understand the controls in place.

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. Much of the data that crosses this network has high integrity and confidentiality requirements and depending on the type of deployment there may also be strong availability requirements. The trust level of this network is heavily dependent on deployment decisions and as such this is not assigned a default level of trust.

Consideration must be taken when managing the users of the system, whether operating or utilizing public or private clouds. The identity service allows for LDAP to be part of the authentication process. Including such systems in your OpenStack deployments may ease user management if integrating into existing systems. Be mindful when utilizing 3rd party clouds to explore authentication options applicable to the installation to help manage and keep user authentication consistent.

Due to the process of passing user names, passwords, and generated tokens between client machines and API endpoints, placing API services behind hardware that performs SSL termination is strongly recommended.

Within cloud components themselves, another component that needs security scrutiny is the hypervisor. In a public cloud, organizations typically do not have control over the choice of hypervisor. (Amazon uses its own particular version of Xen, for example.) In some cases, hypervisors may be vulnerable to a type of attack called "hypervisor breakout" if they are not properly secured. Hypervisor breakout describes the event of a compromised or malicious instance breaking out of the resource controls of the hypervisor and gaining access to the bare metal operating system and hardware resources.

If the security of instances is not considered important, there may not be an issue. In most cases, however, enterprises need to avoid this kind of vulnerability, and the only way to do that is to avoid a situation in which the instances are running on a public cloud. That does not mean that there is a need to own all of the infrastructure on which an OpenStack installation operates; it suggests avoiding situations in which hardware may be shared with others.

There are other services worth considering that provide a bare metal instance instead of a cloud. In other cases, it is possible to replicate a second private cloud by integrating with a private Cloud-as-a-Service deployment, in which an organization does not buy hardware, but also does not share it with other tenants. It is also possible use a provider that hosts a bare-metal "public" cloud instance for which the hardware is dedicated only to one customer, or a provider that offers private Cloud-as-a-Service.

Finally, it is important to realize that each cloud implements services differently. What keeps data secure in one cloud may not do the same in another. Be sure to know the security requirements of every cloud that handles the organization's data or workloads.

More information on OpenStack Security can be found in the *OpenStack Security Guide*.

# Utilization

When it comes to utilization, it is important that the CMP understands what workloads are running, where they are running, and their preferred utilizations. For example, in most cases it is desirable to run as many workloads internally as possible, utilizing other resources only when necessary. On the other hand, situations exist in which the opposite is true. The internal cloud may only be for development and stressing it is undesirable. In most cases, a cost model of various scenarios helps with this decision, however this analysis is heavily influenced by internal priorities. The important thing is the ability to efficiently make those decisions on a programmatic basis.

The Telemetry module (ceilometer) is designed to provide information on the usage of various OpenStack components. There are two limitations to consider: first, if there is to be a large amount of data (for example, if monitoring a large cloud, or a very active one) it is desirable to use a NoSQL back end for Ceilometer, such as MongoDB. Second, when connecting to a non-OpenStack cloud, there will need to be a way to monitor that usage and to provide that monitoring data back to the CMP.

# Performance

Performance is of primary importance in the design of a cloud. When it comes to a hybrid cloud deployment, many of the same issues for multi-site deployments apply, such as network latency between sites. It is also important to think about the speed at which a workload can be spun up in another cloud, and what can be done to reduce the time necessary to accom-

plish that task. That may mean moving data closer to applications, or conversely, applications closer to the data they process. It may mean grouping functionality so that connections that require low latency take place over a single cloud rather than spanning clouds. That may also mean ensuring that the CMP has the intelligence to know which cloud can most efficiently run which types of workloads.

As with utilization, native OpenStack tools are available to assist. Ceilometer can measure performance and, if necessary, the Orchestration module can be used to react to changes in demand by spinning up more resources. It is important to note, however, that Orchestration requires special configurations in the client to enable functioning with solution offerings from Amazon Web Services. When dealing with other types of clouds, it is necessary to rely on the features of the CMP.

# Components

The number and types of native OpenStack components that are available for use is dependent on whether the deployment is exclusively an OpenStack cloud or not. If so, all of the OpenStack components will be available for use, and in many ways the issues that need to be considered will be similar to those that need to be considered for a multi-site deployment.

That said, in any situation in which more than one cloud is being used, at least four OpenStack tools will be considered:

- OpenStack Compute (nova): Regardless of deployment location, hypervisor choice has a direct effect on how difficult it is to integrate with one or more additional clouds. For example, integrating a Hyper-V based OpenStack cloud with Azure will have less compatibility issues than if KVM is used.

- Networking: Whether OpenStack Networking (neutron) or legacy networking (nova-network) is used, the network is one place where integration capabilities need to be understood in order to connect between clouds.

- Telemetry module (ceilometer): Use of Telemetery depends, in large part, on what the other parts of the cloud are using.

- Orchestration module (heat): Similarly, Orchestration can be a valuable tool in orchestrating tasks a CMP decides are necessary in an OpenStack-based cloud.

## Special considerations

Hybrid cloud deployments also involve two more issues that are not common in other situations:

Image portability: Note that, as of the Icehouse release, there is no single common image format that is usable by all clouds. This means that images will need to be converted or recreated when porting between clouds. To make things simpler, launch the smallest and simplest images feasible, installing only what is necessary preferably using a deployment manager such as Chef or Puppet. That means not to use golden images for speeding up the process, however if the same images are being repeatedly deployed it may make more sense to utilize this technique instead of provisioning applications on lighter images each time.

API differences: The most profound issue that cannot be avoided when using a hybrid cloud deployment with more than just OpenStack (or with different versions of OpenStack) is that the APIs needed to perform certain functions are different. The CMP needs to know how to handle all necessary versions. To get around this issue, some implementers build portals to achieve a hybrid cloud environment, but a heavily developer-focused organization will get more use out of a hybrid cloud broker SDK such as jClouds.

# Operational considerations

Hybrid cloud deployments present complex operational challenges. There are several factors to consider that affect the way each cloud is deployed and how users and operators will interact with each cloud. Not every cloud provider implements infrastructure components the same way which may lead to incompatible interactions with workloads or a specific Cloud Management Platform (CMP). Different cloud providers may also offer different levels of integration with competing cloud offerings.

When selecting a CMP, one of the most important aspects to consider is monitoring. Gaining valuable insight into each cloud is critical to gaining a holistic view of all involved clouds. In choosing an existing CMP, determining whether it supports monitoring of all the clouds involved or if compatible APIs are available which can be queried for the necessary information, is vital. Once all the information about each cloud can be gathered and stored in a searchable database, proper actions can be taken on that data offline so workloads will not be impacted.

# Agility

Implementing a hybrid cloud solution can provide application availability across disparate cloud environments and technologies. This availability enables the deployment to survive a complete disaster in any single cloud environment. Each cloud should provide the means to quickly spin up new instances in the case of capacity issues or complete unavailability of a single cloud installation.

# Application readiness

It is important to understand the type of application workloads that will be deployed across the hybrid cloud environment. Enterprise workloads that depend on the underlying infrastructure for availability are not designed to run on OpenStack. Although these types of applications can run on an OpenStack cloud, if the application is not able to tolerate infrastructure failures, it is likely to require significant operator intervention to recover. Cloud workloads, however, are designed with fault tolerance in mind and the SLA of the application is not tied to the underlying infrastructure. Ideally, cloud applications will be designed to recover when entire racks and even data centers full of infrastructure experience an outage.

# Upgrades

OpenStack is a complex and constantly evolving collection of software. Upgrades may be performed to one or more of the cloud environments involved. If a public cloud is involved in the deployment, predicting upgrades may not be possible. Be sure to examine the advertised SLA for any public cloud provider being used. Note that at massive scale, even when dealing with a cloud that offers an SLA with a high percentage of uptime, workloads must be able to recover at short notice.

Similarly, when upgrading private cloud deployments, care must be taken to minimize disruption by making incremental changes and providing a facility to either rollback or continue to roll forward when using a continuous delivery model.

Another consideration is upgrades to the CMP which may need to be completed in coordination with any of the hybrid cloud upgrades. This may be necessary whenever API changes are made in one of the cloud solutions in use to support the new functionality.

## Network Operation Center

When planning the Network Operation Center (NOC) for a hybrid cloud environment, it is important to recognize where control over each piece of infrastructure resides. If a significant portion of the cloud is on externally managed systems, be prepared for situations in which it may not be possible to make changes at all or at the most convenient time. Additionally, situations of conflict may arise in which multiple providers have differing points of view on the way infrastructure must be managed and exposed. This can lead to delays in root cause and analysis where each insists the blame lies with the other provider.

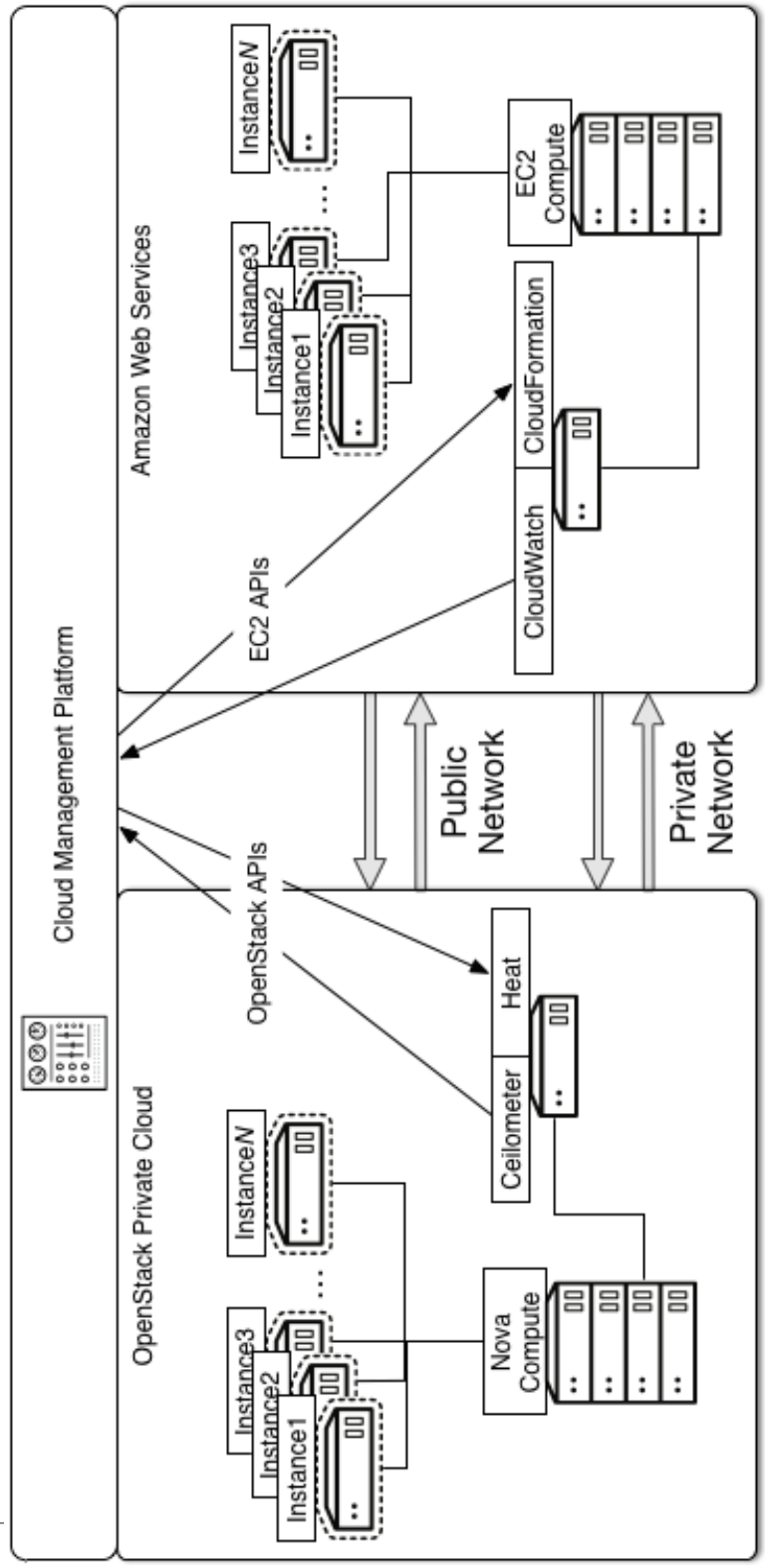
It is important to ensure that the structure put in place enables connection of the networking of both clouds to form an integrated system, keeping in mind the state of handoffs. These handoffs must both be as reliable as possible and include as little latency as possible to ensure the best performance of the overall system.

## Maintainability

Operating hybrid clouds is a situation in which there is a greater reliance on third party systems and processes. As a result of a lack of control of various pieces of a hybrid cloud environment, it is not necessarily possible to guarantee proper maintenance of the overall system. Instead, the user must be prepared to abandon workloads and spin them up again in an improved state. Having a hybrid cloud deployment does, however, provide agility for these situations by allowing the migration of workloads to alternative clouds in response to cloud-specific issues.

# Architecture

Once business and application requirements have been defined, the first step for designing a hybrid cloud solution is to map out the dependencies between the expected workloads and the diverse cloud infrastructures that need to support them. By mapping the applications and the targeted cloud environments, you can architect a solution that enables the broadest compatibility between cloud platforms and minimizes the need to create workarounds and processes to fill identified gaps. Note the evaluation of the monitoring and orchestration APIs available on each cloud platform and the relative levels of support for them in the chosen cloud management platform.


Cloud Management Platform
Amazon Web Services
InstanceN
Instance3
Instance2
Instance1
EC2 Compute
CloudFormation
CloudWatch
EC2 APIs
Public Network
Private Network
OpenStack APIs
OpenStack Private Cloud
Heat
Ceilometer
Nova Compute

# Image portability

The majority of cloud workloads currently run on instances using hypervisor technologies such as KVM, Xen, or ESXi. The challenge is that each of these hypervisors use an image format that is mostly, or not at all, compatible with one another. In a private or hybrid cloud solution, this can be mitigated by standardizing on the same hypervisor and instance image format but this is not always feasible. This is particularly evident if one of the clouds in the architecture is a public cloud that is outside of the control of the designers.

There are conversion tools such as virt-v2v (http://libguestfs.org/virt-v2) and virt-edit (http://libguestfs.org/virt-edit.1.html) that can be used in those scenarios but they are often not suitable beyond very basic cloud instance specifications. An alternative is to build a thin operating system image as the base for new instances. This facilitates rapid creation of cloud instances using cloud orchestration or configuration management tools, driven by the CMP, for more specific templating. Another more expensive option is to use a commercial image migration tool. The issue of image portability is not just for a one time migration. If the intention is to use the multiple cloud for disaster recovery, application diversity or high availability, the images and instances are likely to be moved between the different cloud platforms regularly.

# Upper-layer services

Many clouds offer complementary services over and above the basic compute, network, and storage components. These additional services are often used to simplify the deployment and management of applications on a cloud platform.

Consideration is required to be given to moving workloads that may have upper-layer service dependencies on the source cloud platform to a destination cloud platform that may not have a comparable service. Conversely, the user can implement it in a different way or by using a different technology. For example, moving an application that uses a NoSQL database service such as MongoDB that is delivered as a service on the source cloud, to a destination cloud that does not offer that service or may only use a relational database such as MySQL, could cause difficulties in maintaining the application between the platforms.

There are a number of options that might be appropriate for the hybrid cloud use case:

- Create a baseline of upper-layer services that are implemented across all of the cloud platforms. For platforms that do not support a given service, create a service on top of that platform and apply it to the workloads as they are launched on that cloud. For example, through the *Database Service* for OpenStack (*trove*), OpenStack supports MySQL as a service but not NoSQL databases in production. To either move from or run alongside AWS, a NoSQL workload must use an automation tool, such as the Orchestration module (heat), to recreate the NoSQL database on top of OpenStack.

- Deploy a *Platform-as-a-Service (PaaS)* technology such as Cloud Foundry or OpenShift that abstracts the upper-layer services from the underlying cloud platform. The unit of application deployment and migration is the PaaS and leverages the services of the PaaS and only consumes the base infrastructure services of the cloud platform. The downside to this approach is that the PaaS itself then potentially becomes a source of lock-in.

- Use only the base infrastructure services that are common across all cloud platforms. Use automation tools to create the required upper-layer services which are portable across all cloud platforms. For example, instead of using any database services that are inherent in the cloud platforms, launch cloud instances and deploy the databases on to those instances using scripts or various configuration and application deployment tools.

# Network services

Network services functionality is a significant barrier for multiple cloud architectures. It could be an important factor to assess when choosing a CMP and cloud provider. Considerations are: functionality, security, scalability and *high availability (HA)*. Verification and ongoing testing of the critical features of the cloud endpoint used by the architecture are important tasks.

- Once the network functionality framework has been decided, a minimum functionality test should be designed to confirm that the functionality is in fact compatible. This will ensure testing and functionality persists during and after upgrades. Note that over time, the diverse cloud platforms are likely to de-synchronize if care is not taken to maintain compatibility. This is a particular issue with APIs.

- Scalability across multiple cloud providers may dictate which underlying network framework is chosen for the different cloud providers. It is im-

portant to have the network API functions presented and to verify that the desired functionality persists across all chosen cloud endpoint.

- High availability (HA) implementations vary in functionality and design. Examples of some common methods are active-hot-standby, active-passive and active-active. High availability and a test framework need to be developed to insure that the functionality and limitations are well understood.

- Security considerations, such as how data is secured between client and endpoint and any traffic that traverses the multiple clouds, from eavesdropping to *DoS* activities must be addressed. Business and regulatory requirements dictate the security approach that needs to be taken.

## Data

Replication has been the traditional method for protecting object store implementations. A variety of different implementations have existed in storage architectures. Examples of this are both synchronous and asynchronous mirroring. Most object stores and back-end storage systems have a method for replication that can be implemented at the storage subsystem layer. Object stores also have implemented replication techniques that can be tailored to fit a clouds needs. An organization must find the right balance between data integrity and data availability. Replication strategy may also influence the disaster recovery methods implemented.

Replication across different racks, data centers and geographical regions has led to the increased focus of determining and ensuring data locality. The ability to guarantee data is accessed from the nearest or fastest storage can be necessary for applications to perform well. Examples of this are Hadoop running in a cloud. The user either runs with a native HDFS, when applicable, or on a separate parallel file system such as those provided by Hitachi and IBM. Special consideration should be taken when running embedded object store methods to not cause extra data replication, which can create unnecessary performance issues. Another example of ensuring data locality is by using Ceph. Ceph has a data container abstraction called a pool. Pools can be created with replicas or erasure code. Replica based pools can also have a rule set defined to have data written to a “local” set of hardware which would be the primary access and modification point.

# Prescriptive examples

Multi-cloud environments are typically created to facilitate these use cases:

- Bursting workloads from private to public OpenStack clouds
- Bursting workloads from private to public non-OpenStack clouds
- High availability across clouds (for technical diversity)

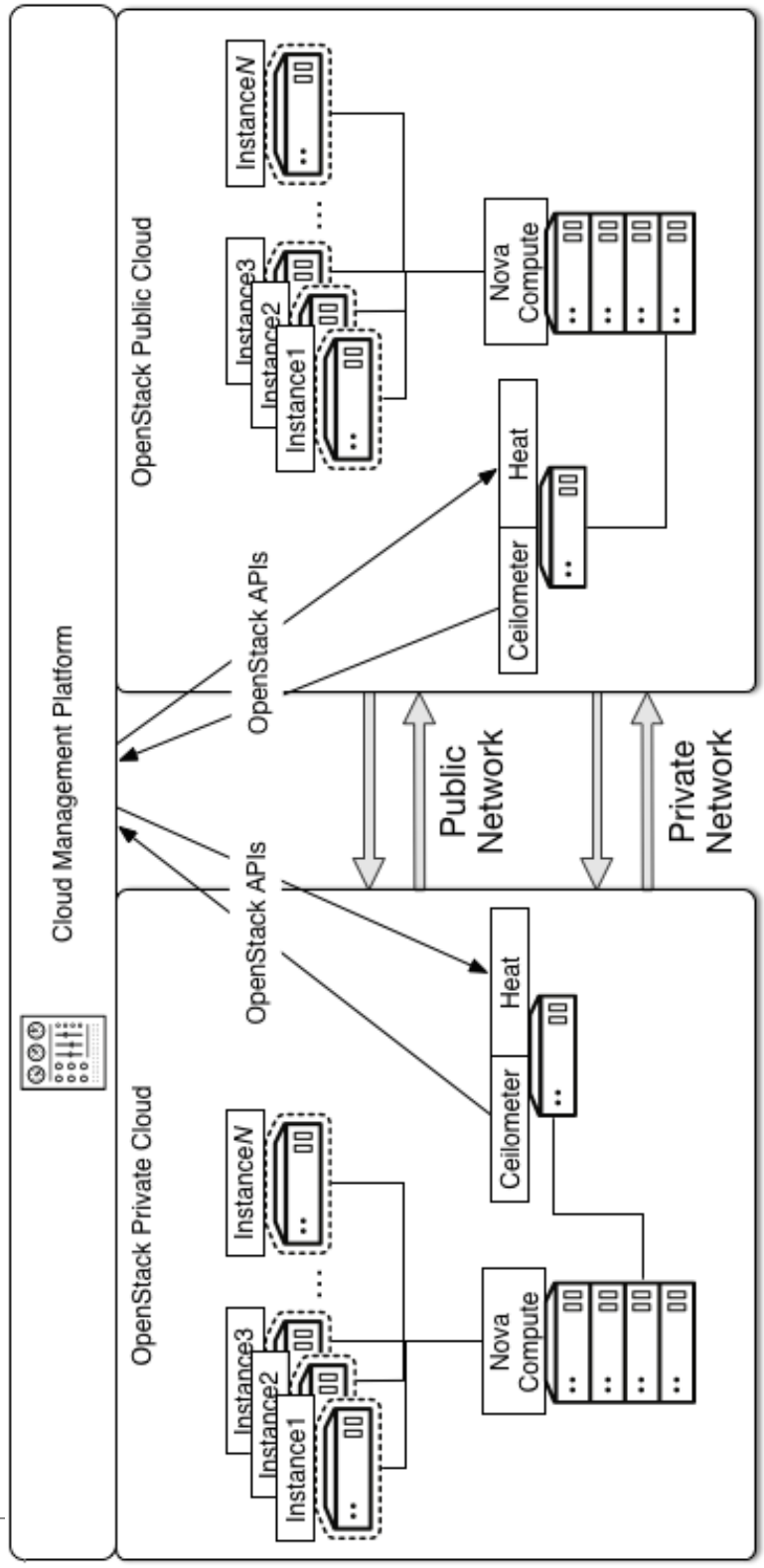
Examples of environments that address each of these use cases will be discussed in this chapter.

Company A's data center is running dangerously low on capacity. The option of expanding the data center will not be possible in the foreseeable future. In order to accommodate the continuously growing need for development resources in the organization, the decision was make use of resource in the public cloud.

The company has an internal cloud management platform that will direct requests to the appropriate cloud, depending on the currently local capacity.

This is a custom in-house application that has been written for this specific purpose.

An example for such a solution is described in the figure below.


Cloud Management Platform
OpenStack Public Cloud
Instance1
Instance2
Instance3
InstanceN
Nova Compute
Ceilometer
Heat
OpenStack APIs
Public Network
Private Network
OpenStack Private Cloud
Instance1
Instance2
Instance3
InstanceN
Nova Compute
Ceilometer
Heat
OpenStack APIs

This example shows two clouds, with a Cloud Management Platform (CMP) connecting them. This guide does not attempt to cover a specific CMP, but describes how workloads are typically orchestrated using the Orchestration and Telemetry services as shown in the diagram above. It is also possibly to connect directly to the other OpenStack APIs with a CMP.

The private cloud is an OpenStack cloud with one or more controllers and one or more compute nodes. It includes metering provided by the Telemetry module. As load increases Telemetry captures this and the information is in turn processed by the CMP. As long as capacity is available, the CMP uses the OpenStack API to call the Orchestration service to create instances on the private cloud in response to user requests. When capacity is not available on the private cloud, the CMP issues a request to the Orchestration service API of the public cloud to create the instance on the public cloud.

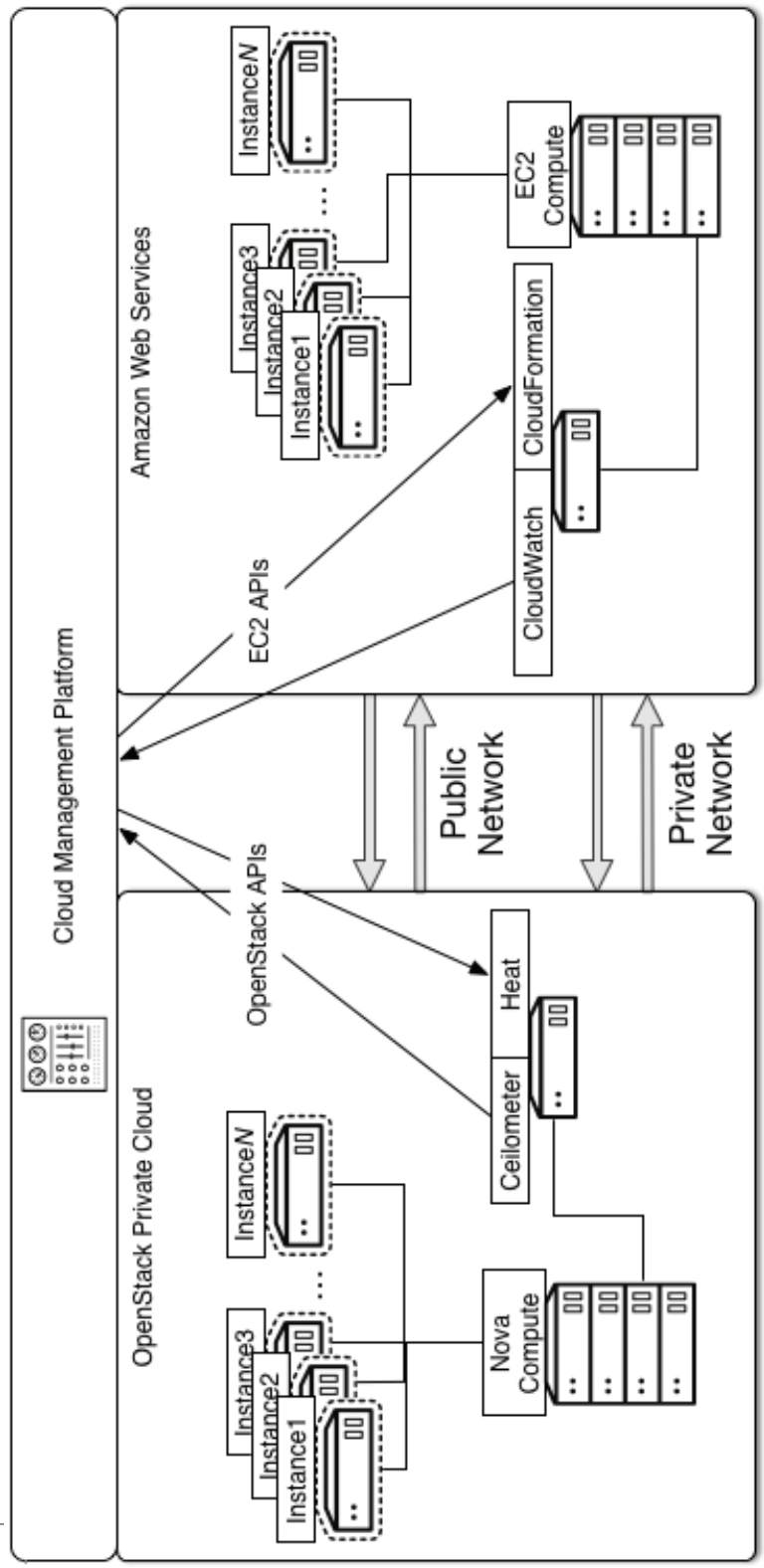
In this example, the whole deployment was not directed to an external public cloud because of the company's fear of lack of resource control and security concerns over control and increased operational expense.

In addition, CompanyA has already established a data center with a substantial amount of hardware, and migrating all the workloads out to a public cloud was not feasible.

## Bursting to a public non-OpenStack cloud

Another common scenario is bursting workloads from the private cloud into a non-OpenStack public cloud such as Amazon Web Services (AWS) to take advantage of additional capacity and scale applications as needed.

For an OpenStack-to-AWS hybrid cloud, the architecture looks similar to the figure below:


Cloud Management Platform
Amazon Web Services
InstanceN
Instance3
Instance2
Instance1
EC2 Compute
CloudFormation
CloudWatch
EC2 APIs
Public Network
Private Network
OpenStack APIs
OpenStack Private Cloud
Heat
Ceilometer
InstanceN
Instance3
Instance2
Instance1
Nova Compute

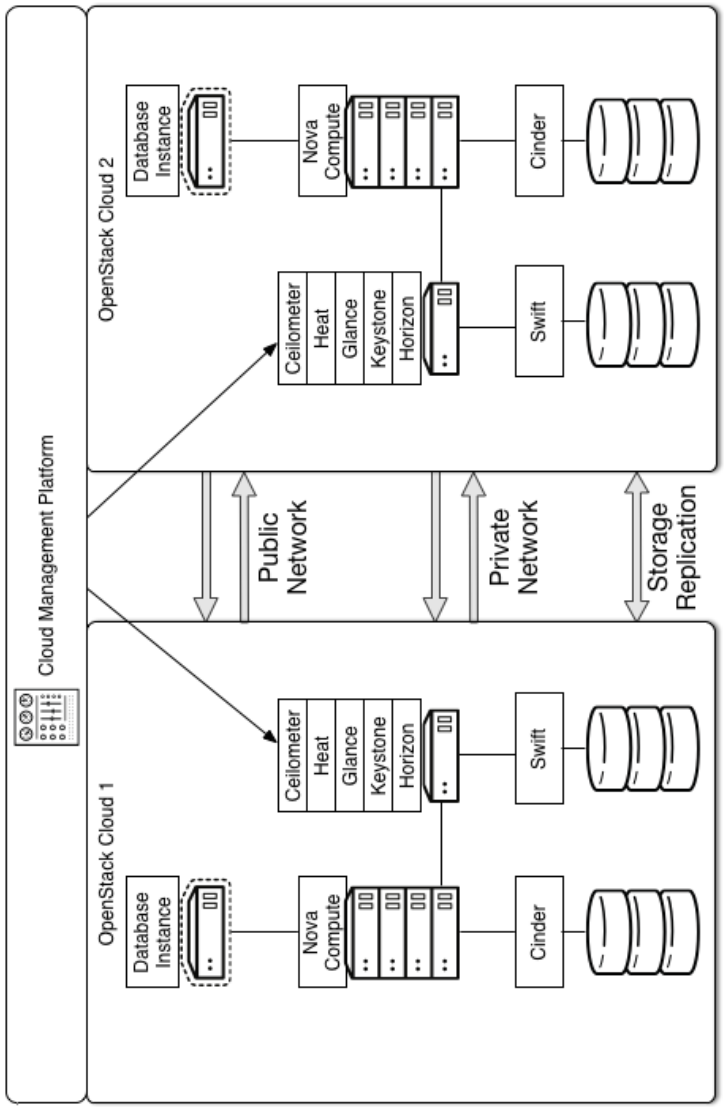
In this scenario company A has an additional requirement in that the developers were already using AWS for some of their work and did not want to change the cloud provider. Primarily due to excessive overhead with network firewall rules that needed to be created and corporate financial procedures that required entering into an agreement with a new provider.

As long as the CMP is capable of connecting an external cloud provider with the appropriate API, the workflow process will remain the same as the previous scenario. The actions the CMP takes such as monitoring load, creating new instances, and so forth are the same, but they would be performed in the public cloud using the appropriate API calls. For example, if the public cloud is Amazon Web Services, the CMP would use the EC2 API to create a new instance and assign an Elastic IP. That IP can then be added to HAProxy in the private cloud, just as it was before. The CMP can also reference AWS-specific tools such as CloudWatch and CloudFormation.

Several open source tool kits for building CMPs are now available that can handle this kind of translation, including ManageIQ, jClouds, and JumpGate.

# High availability/disaster recovery

CompanyA has a requirement to be able to recover from catastrophic failure in their local data center. Some of the workloads currently in use are running on their private OpenStack cloud. Protecting the data involves block storage, object storage, and a database. The architecture is designed to support the failure of large components of the system, yet ensuring that the system will continue to deliver services. While the services remain available to users, the failed components are restored in the background based on standard best practice DR policies. To achieve the objectives, data is replicated to a second cloud, in a geographically distant location. The logical diagram of the system is described in the figure below:




This example includes two private OpenStack clouds connected with a Cloud Management Platform (CMP). The source cloud, OpenStack Cloud 1, includes a controller and at least one instance running MySQL. It also includes at least one block storage volume and one object storage volume so that the data is available to the users at all times. The details of the method for protecting each of these sources of data differs.

The object storage relies on the replication capabilities of the object storage provider. OpenStack Object Storage is enabled so that it creates geographically separated replicas that take advantage of this feature. It is configured so that at least one replica exists in each cloud. In order to make this work a single array spanning both clouds is configured with OpenStack Identity that uses Federated Identity and talks to both clouds, communicating with OpenStack Object Storage through the Swift proxy.

For block storage, the replication is a little more difficult, and involves tools outside of OpenStack itself. The OpenStack Block Storage volume is not set as the drive itself but as a logical object that points to a physical back end. The disaster recovery is configured for Block Storage for synchronous backup for the highest level of data protection, but asynchronous backup could have been set as an alternative that is not as latency sensitive. For asynchronous backup, the Block Storage API makes it possible to export the data and also the metadata of a particular volume, so that it can be moved and replicated elsewhere. More information can be found here: https://blueprints.launchpad.net/cinder/+spec/cinder-backup-volume-metadata-support.

The synchronous backups create an identical volume in both clouds and chooses the appropriate flavor so that each cloud has an identical back end. This was done by creating volumes through the CMP, because the CMP knows to create identical volumes in both clouds. Once this is configured, a solution, involving DRDB, is used to synchronize the actual physical drives.

The database component is backed up using synchronous backups. MySQL does not support geographically diverse replication, so disaster recovery is provided by replicating the file itself. As it is not possible to use object storage as the back end of a database like MySQL, Swift replication was not an option. It was decided not to store the data on another geo-tiered storage system, such as Ceph, as block storage. This would have given another layer of protection. Another option would have been to store the database on an OpenStack Block Storage volume and backing it up just as any other block storage.

# 8. Massively scalable

## Table of Contents



A massively scalable architecture is defined as a cloud implementation that is either a very large deployment, such as one that would be built by a commercial service provider, or one that has the capability to support user requests for large amounts of cloud resources. An example would be an infrastructure in which requests to service 500 instances or more at a time is not uncommon. In a massively scalable infrastructure, such a request is fulfilled without completely consuming all of the available cloud infrastructure resources. While the high capital cost of implementing such a cloud architecture makes it cost prohibitive and is only spearheaded by few organizations, many organizations are planning for massive scalability moving toward the future.

A massively scalable OpenStack cloud design presents a unique set of challenges and considerations. For the most part it is similar to a general purpose cloud architecture, as it is built to address a non-specific range of potential use cases or functions. Typically, it is rare that massively scalable clouds are designed or specialized for particular workloads. Like the general purpose cloud, the massively scalable cloud is most often built as a platform for a variety of workloads. Massively scalable OpenStack clouds are generally built as commercial public cloud offerings since single private organizations rarely have the resources or need for this scale.

Services provided by a massively scalable OpenStack cloud will include:

- Virtual-machine disk image library
- Raw block storage
- File or object storage
- Firewall functionality
- Load balancing functionality

- Private (non-routable) and public (floating) IP addresses
- Virtualized network topologies
- Software bundles
- Virtual compute resources

Like a general purpose cloud, the instances deployed in a massively scalable OpenStack cloud will not necessarily use any specific aspect of the cloud offering (compute, network, or storage). As the cloud grows in scale, the scale of the number of workloads can cause stress on all of the cloud components. Additional stresses are introduced to supporting infrastructure including databases and message brokers. The architecture design for such a cloud must account for these performance pressures without negatively impacting user experience.

# User requirements

More so than other scenarios, defining user requirements for a massively scalable OpenStack design architecture dictates approaching the design from two different, yet sometimes opposing, perspectives: the cloud user, and the cloud operator. The expectations and perceptions of the consumption and management of resources of a massively scalable OpenStack cloud from the user point of view is distinctly different from that of the cloud operator.

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.
- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.
- Data compliance policies governing certain types of information needs to reside in certain locations due to regular issues and, more importantly, cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Reg-

ulatory Authority in the United States. Consult a local regulatory body for more information.

# User requirements

Massively scalable OpenStack clouds have the following user requirements:

- The cloud user expects repeatable, dependable, and deterministic processes for launching and deploying cloud resources. This could be delivered through a web-based interface or publicly available API endpoints. All appropriate options for requesting cloud resources need to be available through some type of user interface, a command-line interface (CLI), or API endpoints.
- Cloud users expect a fully self-service and on-demand consumption model. When an OpenStack cloud reaches the "massively scalable" size, it means it is expected to be consumed "as a service" in each and every way.
- For a user of a massively scalable OpenStack public cloud, there will be no expectations for control over security, performance, or availability. Only SLAs related to uptime of API services are expected, and very basic SLAs expected of services offered. The user understands it is his or her responsibility to address these issues on their own. The exception to this expectation is the rare case of a massively scalable cloud infrastructure built for a private or government organization that has specific requirements.

As might be expected, the cloud user requirements or expectations that determine the design are all focused on the consumption model. The user expects to be able to easily consume cloud resources in an automated and deterministic way, without any need for knowledge of the capacity, scalability, or other attributes of the cloud's underlying infrastructure.

# Operator requirements

Whereas the cloud user should be completely unaware of the underlying infrastructure of the cloud and its attributes, the operator must be able to build and support the infrastructure, as well as how it needs to operate at scale. This presents a very demanding set of requirements for building such a cloud from the operator's perspective:

- First and foremost, everything must be capable of automation. From the deployment of new hardware, compute hardware, storage hardware, or networking hardware, to the installation and configuration of the

supporting software, everything must be capable of being automated. Manual processes will not suffice in a massively scalable OpenStack design architecture.

- The cloud operator requires that capital expenditure (CapEx) is minimized at all layers of the stack. Operators of massively scalable OpenStack clouds require the use of dependable commodity hardware and freely available open source software components to reduce deployment costs and operational expenses. Initiatives like OpenCompute (more information available at http://www.opencompute.org) provide additional information and pointers. To cut costs, many operators sacrifice redundancy. For example, redundant power supplies, redundant network connections, and redundant rack switches.

- Companies operating a massively scalable OpenStack cloud also require that operational expenditures (OpEx) be minimized as much as possible. It is recommended that cloud-optimized hardware is a good approach when managing operational overhead. Some of the factors that need to be considered include power, cooling, and the physical design of the chassis. It is possible to customize the hardware and systems so they are optimized for this type of workload because of the scale of these implementations.

- Massively scalable OpenStack clouds require extensive metering and monitoring functionality to maximize the operational efficiency by keeping the operator informed about the status and state of the infrastructure. This includes full scale metering of the hardware and software status. A corresponding framework of logging and alerting is also required to store and allow operations to act upon the metrics provided by the metering and monitoring solution(s). The cloud operator also needs a solution that uses the data provided by the metering and monitoring solution to provide capacity planning and capacity trending analysis.

- A massively scalable OpenStack cloud will be a multi-site cloud. Therefore, the user-operator requirements for a multi-site OpenStack architecture design are also applicable here. This includes various legal requirements for data storage, data placement, and data retention; other jurisdictional legal or compliance requirements; image consistency-availability; storage replication and availability (both block and file/object storage); and authentication, authorization, and auditing (AAA), just to name a few. Refer to the Chapter 6, “Multi-site” [135] for more details on requirements and considerations for multi-site OpenStack clouds.

- Considerations around physical facilities such as space, floor weight, rack height and type, environmental considerations, power usage and power

usage efficiency (PUE), and physical security must also be addressed by the design architecture of a massively scalable OpenStack cloud.

# Technical considerations

Converting an existing OpenStack environment that was designed for a different purpose to be massively scalable is a formidable task. When building a massively scalable environment from the ground up, make sure the initial deployment is built with the same principles and choices that apply as the environment grows. For example, a good approach is to deploy the first site as a multi-site environment. This allows the same deployment and segregation methods to be used as the environment grows to separate locations across dedicated links or wide area networks. In a hyperscale cloud, scale trumps redundancy. Applications must be modified with this in mind, relying on the scale and homogeneity of the environment to provide reliability rather than redundant infrastructure provided by non-commodity hardware solutions.

## Infrastructure segregation

Fortunately, OpenStack services are designed to support massive horizontal scale. Be aware that this is not the case for the entire supporting infrastructure. This is particularly a problem for the database management systems and message queues used by the various OpenStack services for data storage and remote procedure call communications.

Traditional clustering techniques are typically used to provide high availability and some additional scale for these environments. In the quest for massive scale, however, additional steps need to be taken to relieve the performance pressure on these components to prevent them from negatively impacting the overall performance of the environment. It is important to make sure that all the components are in balance so that, if and when the massively scalable environment fails, all the components are at, or close to, maximum capacity.

Regions are used to segregate completely independent installations linked only by an Identity and Dashboard (optional) installation. Services are installed with separate API endpoints for each region, complete with separate database and queue installations. This exposes some awareness of the environment's fault domains to users and gives them the ability to ensure some degree of application resiliency while also imposing the requirement to specify which region their actions must be applied to.

Environments operating at massive scale typically need their regions or sites subdivided further without exposing the requirement to specify the failure domain to the user. This provides the ability to further divide the installation into failure domains while also providing a logical unit for maintenance and the addition of new hardware. At hyperscale, instead of adding single compute nodes, administrators may add entire racks or even groups of racks at a time with each new addition of nodes exposed via one of the segregation concepts mentioned herein.

*Cells* provide the ability to subdivide the compute portion of an OpenStack installation, including regions, while still exposing a single endpoint. In each region an API cell is created along with a number of compute cells where the workloads actually run. Each cell gets its own database and message queue setup (ideally clustered), providing the ability to subdivide the load on these subsystems, improving overall performance.

Within each compute cell a complete compute installation is provided, complete with full database and queue installations, scheduler, conductor, and multiple compute hosts. The cells scheduler handles placement of user requests from the single API endpoint to a specific cell from those available. The normal filter scheduler then handles placement within the cell.

The downside of using cells is that they are not well supported by any of the OpenStack services other than Compute. Also, they do not adequately support some relatively standard OpenStack functionality such as security groups and host aggregates. Due to their relative newness and specialized use, they receive relatively little testing in the OpenStack gate. Despite these issues, however, cells are used in some very well known OpenStack installations operating at massive scale including those at CERN and Rackspace.

## Host aggregates

Host aggregates enable partitioning of OpenStack Compute deployments into logical groups for load balancing and instance distribution. Host aggregates may also be used to further partition an availability zone. Consider a cloud which might use host aggregates to partition an availability zone into groups of hosts that either share common resources, such as storage and network, or have a special property, such as trusted computing hardware. Host aggregates are not explicitly user-targetable; instead they are implicitly targeted via the selection of instance flavors with extra specifications that map to host aggregate metadata.

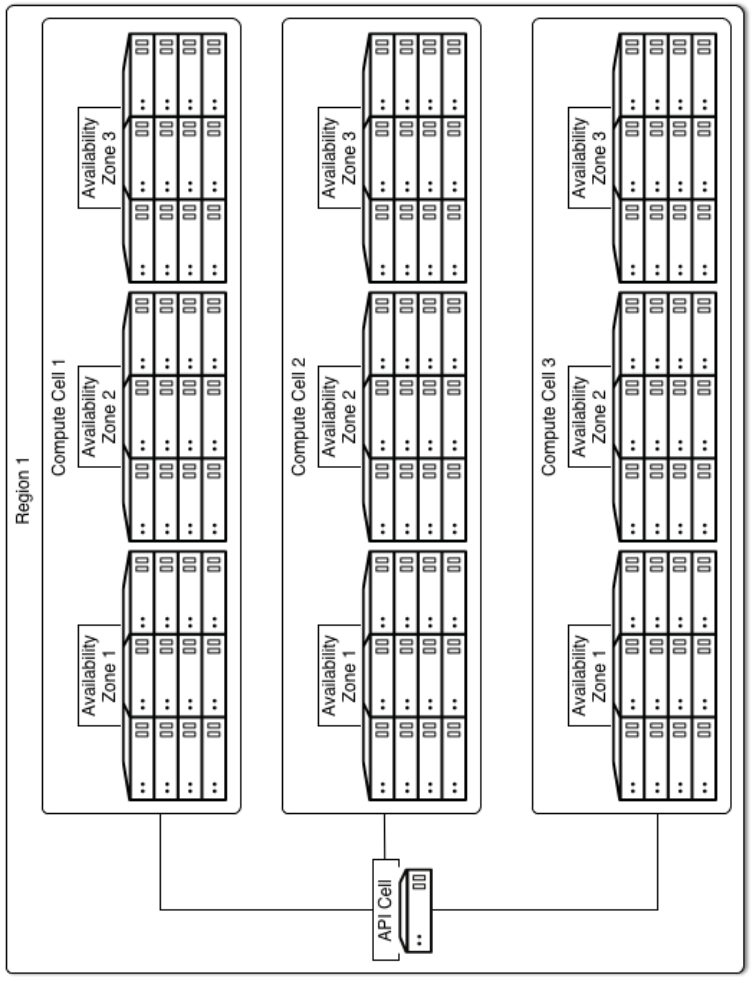
# Availability zones

Availability zones provide another mechanism for subdividing an installation or region. They are, in effect, host aggregates that are exposed for (optional) explicit targeting by users.

Unlike cells, they do not have their own database server or queue broker but simply represent an arbitrary grouping of compute nodes. Typically, grouping of nodes into availability zones is based on a shared failure domain based on a physical characteristic such as a shared power source, physical network connection, and so on. Availability zones are exposed to the user because they can be targeted; however, users are not required to target them. An alternate approach is for the operator to set a default availability zone to schedule instances to other than the default availability zone of nova.

# Segregation example

In this example the cloud is divided into two regions, one for each site, with two availability zones in each based on the power layout of the data centers. A number of host aggregates have also been defined to allow targeting of virtual machine instances using flavors, that require special capabilities shared by the target hosts such as SSDs, 10 GbE networks, or GPU cards.




# Operational considerations

In order to run at massive scale, it is important to plan on the automation of as many of the operational processes as possible. Automation includes the configuration of provisioning, monitoring and alerting systems. Part of the automation process includes the capability to determine when human intervention is required and who should act. The objective is to increase the ratio of operational staff to running systems as much as possible to reduce maintenance costs. In a massively scaled environment, it is impossible for staff to give each system individual care.

Configuration management tools such as Puppet or Chef allow operations staff to categorize systems into groups based on their role and thus create configurations and system states that are enforced through the provisioning system. Systems that fall out of the defined state due to errors or failures are quickly removed from the pool of active nodes and replaced.

At large scale the resource cost of diagnosing individual systems that have failed is far greater than the cost of replacement. It is more economical to immediately replace the system with a new system that can be provisioned and configured automatically and quickly brought back into the pool of active nodes. By automating tasks that are labor-intensive, repetitive, and critical to operations with automation, cloud operations teams are able to be managed more efficiently because fewer resources are needed for these babysitting tasks. Administrators are then free to tackle tasks that cannot be easily automated and have longer-term impacts on the business such as capacity planning.

# The bleeding edge

Running OpenStack at massive scale requires striking a balance between stability and features. For example, it might be tempting to run an older stable release branch of OpenStack to make deployments easier. However, when running at massive scale, known issues that may be of some concern or only have minimal impact in smaller deployments could become pain points at massive scale. If the issue is well known, in many cases, it may be resolved in more recent releases. The OpenStack community can help resolve any issues reported by the applying the collective expertise of the OpenStack developers.

When issues crop up, the number of organizations running at a similar scale is a relatively tiny proportion of the OpenStack community, therefore it is important to share these issues with the community and be a vocal advocate for resolving them. Some issues only manifest when operating at large scale and the number of organizations able to duplicate and validate an issue is small, so it will be important to document and dedicate resources to their resolution.

In some cases, the resolution to the problem is ultimately to deploy a more recent version of OpenStack. Alternatively, when the issue needs to be resolved in a production environment where rebuilding the entire environment is not an option, it is possible to deploy just the more recent separate underlying components required to resolve issues or gain significant performance improvements. At first glance, this could be perceived as poten-

tially exposing the deployment to increased risk to and instability. However, in many cases it could be an issue that has not been discovered yet.

It is advisable to cultivate a development and operations organization that is responsible for creating desired features, diagnose and resolve issues, and also build the infrastructure for large scale continuous integration tests and continuous deployment. This helps catch bugs early and make deployments quicker and less painful. In addition to development resources, the recruitment of experts in the fields of message queues, databases, distributed systems, and networking, cloud and storage is also advisable.

## Growth and capacity planning

An important consideration in running at massive scale is projecting growth and utilization trends to plan capital expenditures for the near and long term. Utilization metrics for compute, network, and storage as well as a historical record of these metrics are required. While securing major anchor tenants can lead to rapid jumps in the utilization rates of all resources, the steady adoption of the cloud inside an organizations or by public consumers in a public offering will also create a steady trend of increased utilization.

## Skills and training

Projecting growth for storage, networking, and compute is only one aspect of a growth plan for running OpenStack at massive scale. Growing and nurturing development and operational staff is an additional consideration. Sending team members to OpenStack conferences, meetup events, and encouraging active participation in the mailing lists and committees is a very important way to maintain skills and forge relationships in the community. A list of OpenStack training providers in the marketplace can be found here: http://www.openstack.org/marketplace/training/.

# 9. Specialized cases

## Table of Contents



Although most OpenStack architecture designs fall into one of the seven major scenarios outlined in other sections (compute focused, network focused, storage focused, general purpose, multi-site, hybrid cloud, and massively scalable), there are a few other use cases that are unique enough they can't be neatly categorized into one of the other major sections. This section discusses some of these unique use cases with some additional details and design considerations for each use case:

- Specialized Networking: This describes running networking-oriented software that may involve reading packets directly from the wire or participating in routing protocols.

- Software-defined networking (SDN): This use case details both running an SDN controller from within OpenStack as well as participating in a software-defined network.

- *Desktop-as-a-Service*: This is for organizations that want to run a virtualized desktop environment on a cloud. This can apply to private or public clouds.

- OpenStack on OpenStack: Some organizations are finding that it makes technical sense to build a multi-tiered cloud by running OpenStack on top of an OpenStack installation.

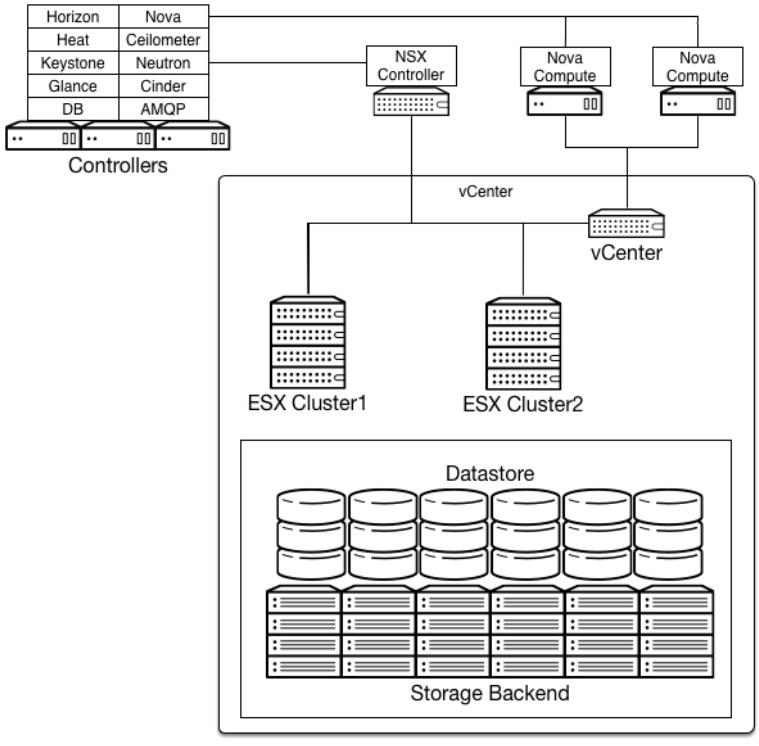
- Specialized hardware: Some highly specialized situations will require the use of specialized hardware devices from within the OpenStack environment.

# Multi-hypervisor example

A financial company requires a migration of its applications from a traditional virtualized environment to an API driven, orchestrated environment. A number of their applications have strict support requirements which limit what hypervisors they are supported on, however the rest do not have such restrictions and do not need the same features. Because of these requirements, the overall target environment needs multiple hypervisors.

The current environment consists of a vSphere environment with 20 VMware ESXi hypervisors supporting 300 instances of various sizes. Approximately 50 of the instances must be run on ESXi but the rest have more flexible requirements.

The company has decided to bring the management of the overall system into a common platform provided by OpenStack.




The approach is to run a host aggregate consisting of KVM hypervisors for the general purpose instances and a separate host aggregate for instances requiring ESXi. This way, workloads that must be run on ESXi can be tar-

geted at those hypervisors, but the rest can be targeted at the KVM hypervisors.

Images in the OpenStack Image Service have particular hypervisor metadata attached so that when a user requests a certain image, the instance will spawn on the relevant aggregate. Images for ESXi are stored in VMDK format. QEMU disk images can be converted to VMDK, VMFS Flat Disks, which includes thin, thick, zeroed-thick, and eager-zeroed-thick. Note that once a VMFS thin disk is exported from VMFS to a non-VMFS location, like the OpenStack Image Service, it becomes a preallocated flat disk. This impacts the transfer time from the OpenStack Image Service to the data store when the full preallocated flat disk, rather than the thin disk, must be transferred.

This example has the additional complication that, rather than being spawned directly on a hypervisor simply by calling a specific host aggregate using the metadata of the image, the VMware host aggregate compute nodes communicate with vCenter which then requests that the instance be scheduled to run on an ESXi hypervisor. As of the Icehouse release, this functionality requires that VMware Distributed Resource Scheduler (DRS) be enabled on a cluster and set to "Fully Automated".

Due to the DRS requirement, note that vSphere requires shared storage (DRS uses vMotion, which requires shared storage). The solution uses shared storage to provide Block Storage capabilities to the KVM instances while also providing the storage for vSphere. The environment uses a dedicated data network to provide this functionality, therefore the compute hosts should have dedicated NICs to support this dedicated traffic. vSphere supports the use of OpenStack Block Storage to present storage from a VMFS datastore to an instance, so the use of Block Storage in this architecture supports both hypervisors.

In this case, network connectivity is provided by OpenStack Networking with the VMware NSX plug-in driver configured. Alternatively, the system could use legacy networking (nova-network), which is supported by both hypervisors used in this design, but has limitations. Specifically, security groups are not supported on vSphere with legacy networking. With VMware NSX as part of the design, however, when a user launches an instance within either of the host aggregates, the instances are attached to appropriate network overlay-based logical networks as defined by the user.

Note that care must be taken with this approach, as there are design considerations around the OpenStack Compute integration. When using vSphere with OpenStack, the nova-compute service that is configured to

communicate with vCenter shows up as a single large hypervisor representing the entire ESXi cluster (multiple instances of nova-compute can be run to represent multiple ESXi clusters or to connect to multiple vCenter servers). If the process running the nova-compute service crashes, the connection to that particular vCenter Server-and any ESXi clusters behind it-are severed and it will not be possible to provision more instances on that vCenter, despite the fact that vSphere itself could be configured for high availability. Therefore, it is important to monitor the nova-compute service that connects to vSphere for any disruptions.

# Specialized networking example

Some applications that interact with a network require more specialized connectivity. Applications such as a looking glass require the ability to connect to a BGP peer, or route participant applications may need to join a network at a layer 2 level.

## Challenges

Connecting specialized network applications to their required resources alters the design of an OpenStack installation. Installations that rely on overlay networks are unable to support a routing participant, and may also block layer-2 listeners.

## Possible solutions

Deploying an OpenStack installation using OpenStack Networking with a provider network will allow direct layer-2 connectivity to an upstream networking device. This design provides the layer-2 connectivity required to communicate via Intermediate System-to-Intermediate System (ISIS) protocol or to pass packets controlled via an OpenFlow controller. Using the multiple layer-2 plug-in with an agent such as *Open vSwitch* would allow a private connection through a VLAN directly to a specific port in a layer-3 device. This would allow a BGP point to point link to exist that will join the autonomous system. Avoid using layer-3 plug-ins as they will divide the broadcast domain and prevent router adjacencies from forming.

# Software-defined networking

Software-defined networking (SDN) is the separation of the data plane and control plane. SDN has become a popular method of managing and

controlling packet flows within networks. SDN uses overlays or directly controlled layer-2 devices to determine flow paths, and as such presents challenges to a cloud environment. Some designers may wish to run their controllers within an OpenStack installation. Others may wish to have their installations participate in an SDN-controlled network.

## Challenges

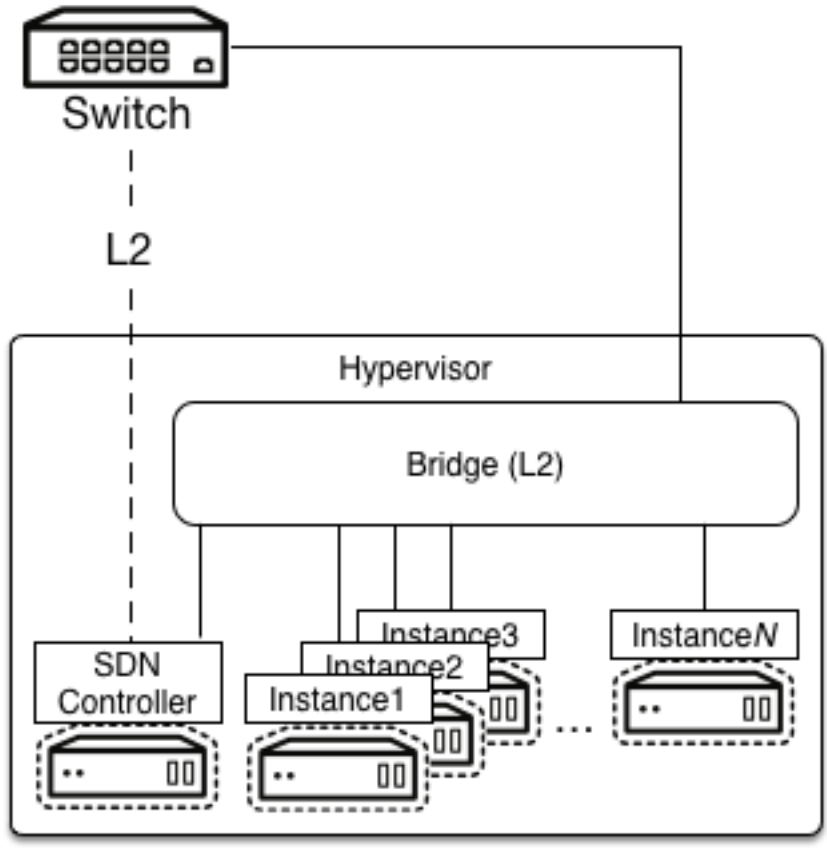
SDN is a relatively new concept that is not yet standardized, so SDN systems come in a variety of different implementations. Because of this, a truly prescriptive architecture is not feasible. Instead, examine the differences between an existing or intended OpenStack design and determine where the potential conflict and gaps can be found.

## Possible solutions

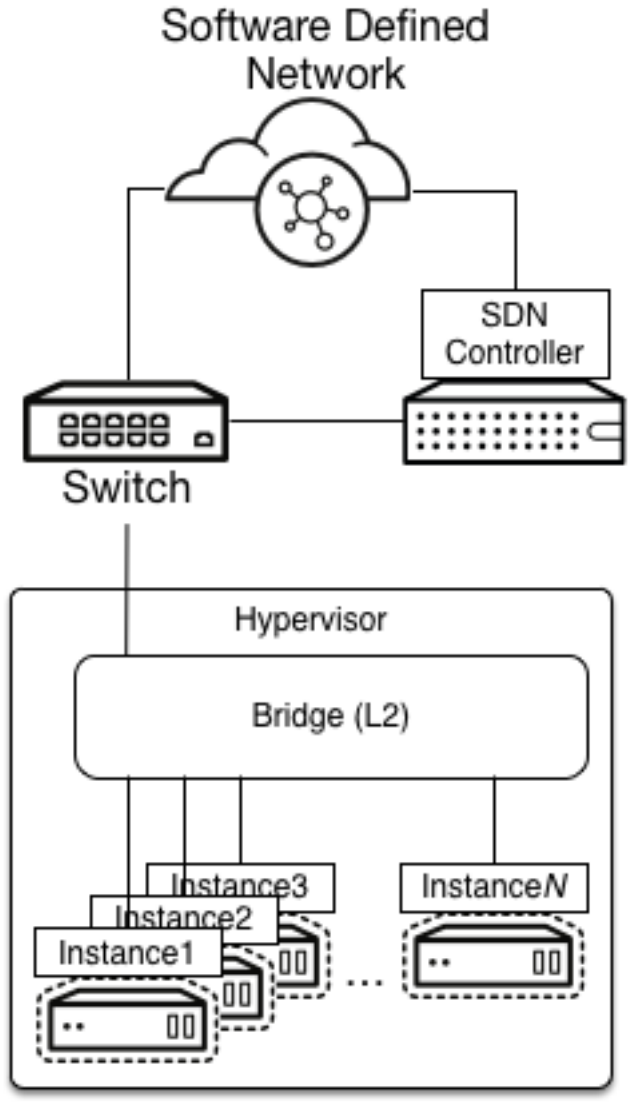
If an SDN implementation requires layer-2 access because it directly manipulates switches, then running an overlay network or a layer-3 agent may not be advisable. If the controller resides within an OpenStack installation, it may be necessary to build an ML2 plug-in and schedule the controller instances to connect to tenant VLANs that then talk directly to the switch hardware. Alternatively, depending on the external device support, use a tunnel that terminates at the switch hardware itself.

## Diagram

OpenStack hosted SDN controller:




OpenStack participating in an SDN controller network:




# Desktop-as-a-Service

Virtual Desktop Infrastructure (VDI) is a service that hosts user desktop environments on remote servers. This application is very sensitive to network latency and requires a high performance compute environment. Traditionally these types of environments have not been put on cloud environments because few clouds are built to support such a demanding workload that is so exposed to end users. Recently, as cloud environments become more

robust, vendors are starting to provide services that allow virtual desktops to be hosted in the cloud. In the not too distant future, OpenStack could be used as the underlying infrastructure to run a virtual infrastructure environment, either in-house or in the cloud.

## Challenges

Designing an infrastructure that is suitable to host virtual desktops is a very different task to that of most virtual workloads. The infrastructure will need to be designed, for example:

- Boot storms: What happens when hundreds or thousands of users log in during shift changes, affects the storage design.
- The performance of the applications running in these virtual desktops
- Operating system and compatibility with the OpenStack hypervisor

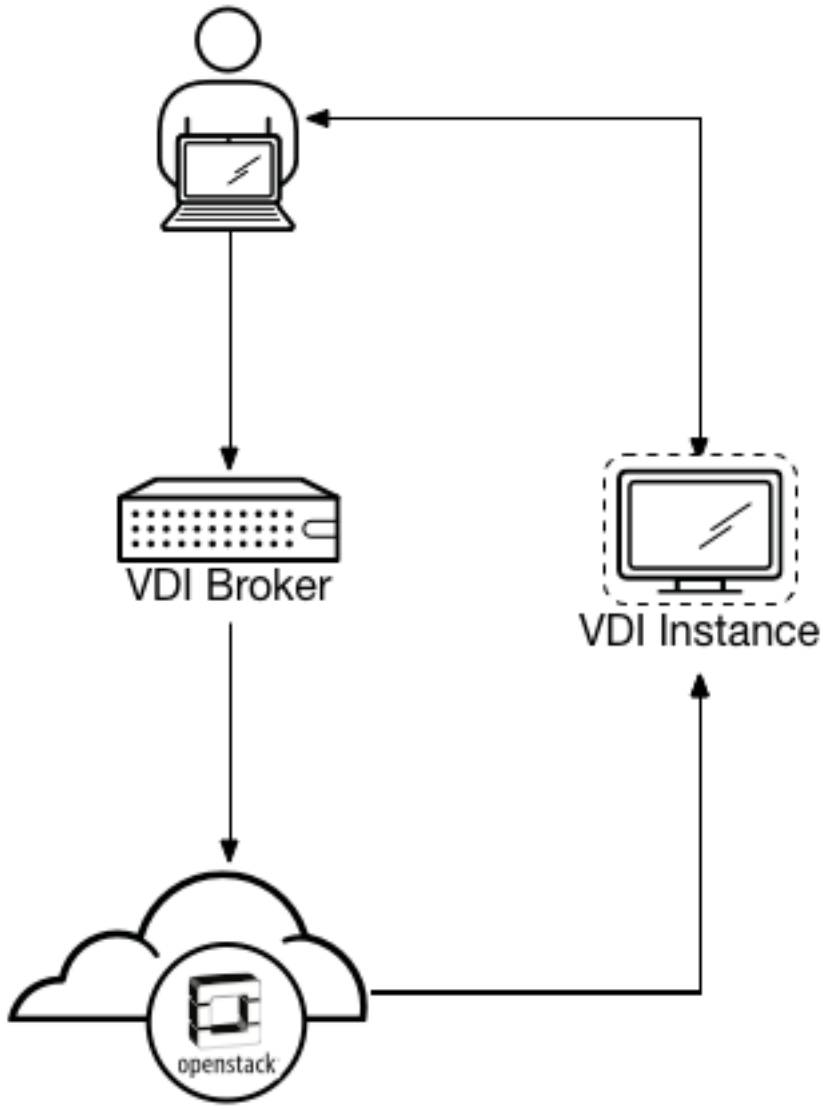
## Broker

The connection broker is a central component of the architecture that determines which remote desktop host will be assigned or connected to the user. The broker is often a full-blown management product allowing for the automated deployment and provisioning of remote desktop hosts.

## Possible solutions

There are a number of commercial products available today that provide such a broker solution but nothing that is native in the OpenStack project. Not providing a broker is also an option, but managing this manually would not suffice as a large scale, enterprise solution.

## Diagram




# OpenStack on OpenStack

In some cases it is necessary to run OpenStack nested on top of another OpenStack cloud. This scenario allows for complete OpenStack cloud environments to be managed and provisioned on instances running on hypervisors and servers controlled by the underlying OpenStack cloud. Public

cloud providers can use this technique to effectively manage the upgrade and maintenance process on complete OpenStack-based clouds. Developers and those testing OpenStack can also use the guidance to provision their own OpenStack environments on available OpenStack Compute resources, whether public or private.

## Challenges

The network aspect of deploying a nested cloud is the most complicated aspect of this architecture. When using VLANs, these will need to be exposed to the physical ports on which the undercloud runs, as the bare metal cloud owns all the hardware, but they also need to be exposed to the nested levels as well. Alternatively, network overlay technologies can be used on the overcloud (the OpenStack cloud running on OpenStack) to provide the required software defined networking for the deployment.

## Hypervisor

A key question to address in this scenario is the decision about which approach should be taken to provide a nested hypervisor in OpenStack. This decision influences which operating systems can be used for the deployment of the nested OpenStack deployments.

## Possible solutions: deployment

Deployment of a full stack can be challenging but this difficulty can be readily be mitigated by creating a Heat template to deploy the entire stack or a configuration management system. Once the Heat template is created, deploying additional stacks will be a trivial thing and can be performed in an automated fashion.

The OpenStack-on-OpenStack project (*TripleO*) addresses this issue—currently, however, the project does not completely cover nested stacks. For more information, see https://wiki.openstack.org/wiki/TripleO.

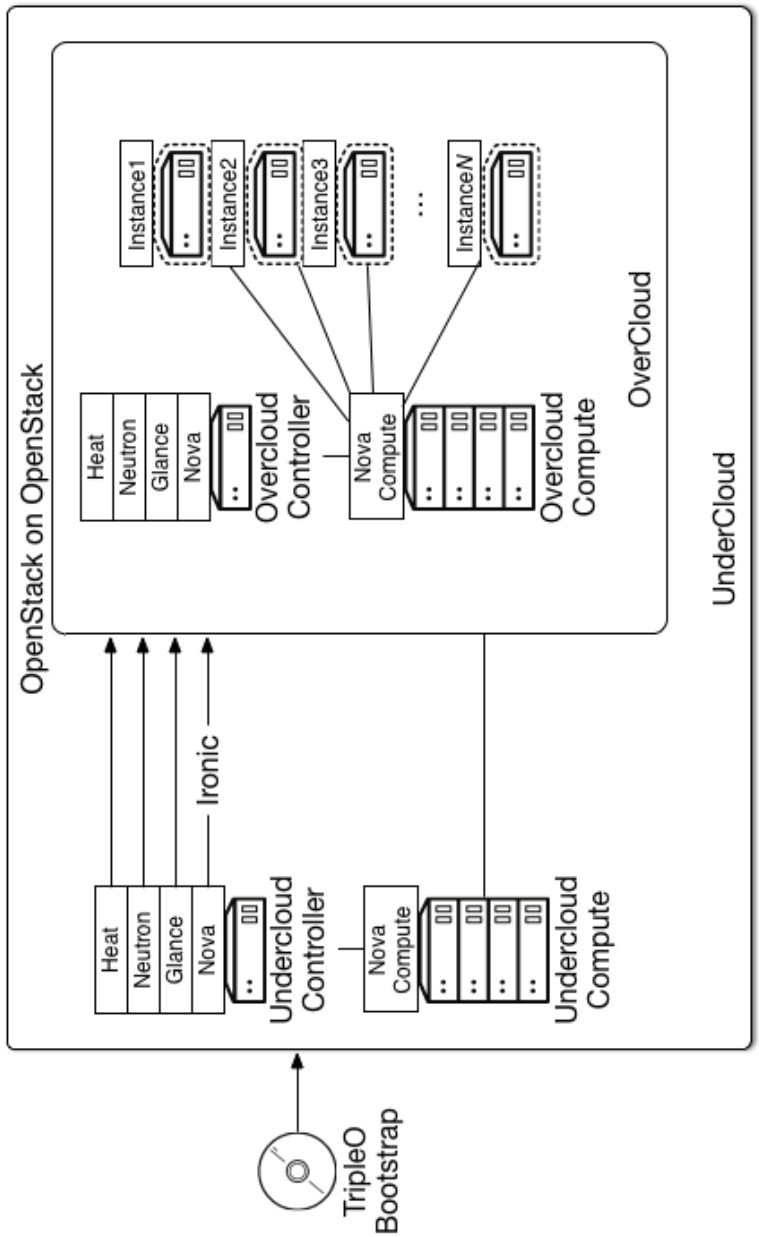
## Possible solutions: hypervisor

In the case of running TripleO, the underlying OpenStack cloud deploys the Compute nodes as bare-metal. OpenStack would then be deployed on these Compute bare-metal servers with the appropriate hypervisor, such as KVM.

In the case of running smaller OpenStack clouds for testing purposes, and performance would not be a critical factor, QEMU can be utilized instead.

It is also possible to run a KVM hypervisor in an instance (see http://davejingtian.org/2014/03/30/nested-kvm-just-for-fun/), though this is not a supported configuration, and could be a complex solution for such a use case.

# Diagram


OpenStack on OpenStack
Instance1
Instance2
Instance3
...
InstanceN
Heat
Neutron
Glance
Nova
Overcloud Controller
Nova Compute
Overcloud Compute
OverCloud
UnderCloud
Ironic
Undercloud Controller
Undercloud Compute
TripleO Bootstrap

# Specialized hardware

Certain workloads require specialized hardware devices that are either difficult to virtualize or impossible to share. Applications such as load balancers, highly parallel brute force computing, and direct to wire networking may need capabilities that basic OpenStack components do not provide.

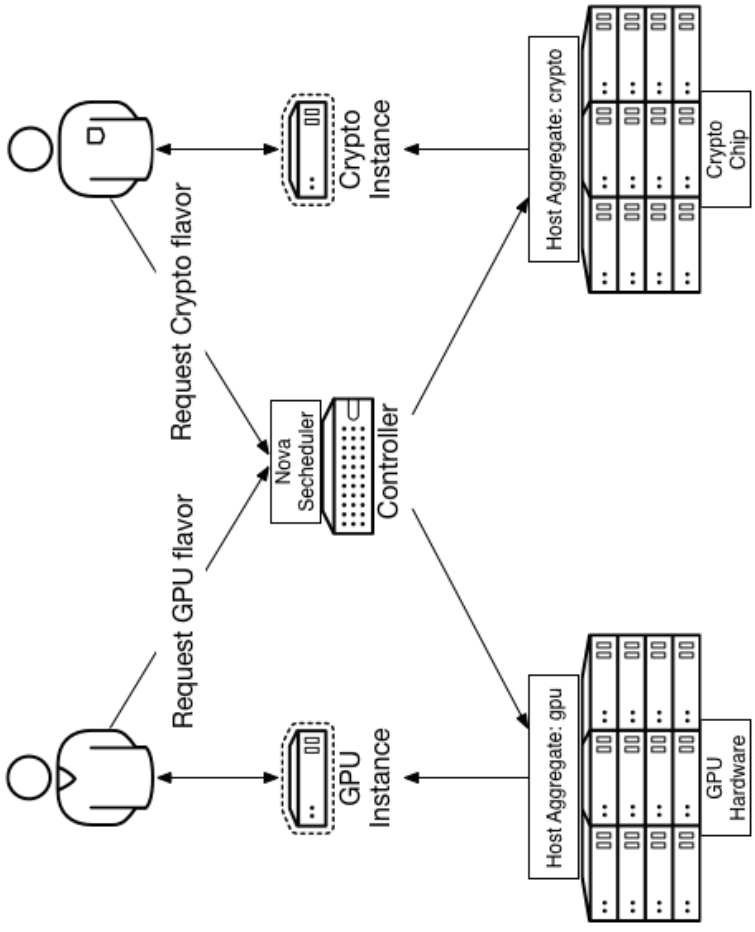
## Challenges

Some applications need access to hardware devices to either improve performance or provide capabilities that are not virtual CPU, RAM, network or storage. These can be a shared resource, such as a cryptography processor, or a dedicated resource such as a Graphics Processing Unit. OpenStack has ways of providing some of these, while others may need extra work.

## Solutions

In order to provide cryptography offloading to a set of instances, it is possible to use Image Service configuration options to assign the cryptography chip to a device node in the guest. The *OpenStack Command Line Reference* contains further information on configuring this solution in the chapter Image Service property keys , but it allows all guests using the configured images to access the hypervisor cryptography device.

If direct access to a specific device is required, it can be dedicated to a single instance per hypervisor through the use of PCI pass-through. The OpenStack administrator needs to define a flavor that specifically has the PCI device in order to properly schedule instances. More information regarding PCI pass-through, including instructions for implementing and using it, is available at https://wiki.openstack.org/wiki/Pci_passthrough.


Request Crypto flavor
Request GPU flavor
Nova Secheduler
Controller
Crypto Instance
GPU Instance
Host Aggregate: crypto
Crypto Chip
Host Aggregate: gpu
GPU Hardware

# 10. References

Data Protection framework of the European Union: Guidance on Data Protection laws governed by the EU.

Depletion of IPv4 Addresses: describing how IPv4 addresses and the migration to IPv6 is inevitable.

Ethernet Switch Reliability: Research white paper on Ethernet Switch reliability.

Financial Industry Regulatory Authority: Requirements of the Financial Industry Regulatory Authority in the USA.

Image Service property keys: Glance API property keys allows the administrator to attach custom characteristics to images.

LibGuestFS Documentation: Official LibGuestFS documentation.

Logging and Monitoring: Official OpenStack Operations documentation.

ManageIQ Cloud Management Platform: An Open Source Cloud Management Platform for managing multiple clouds.

N-Tron Network Availability: Research white paper on network availability.

Nested KVM: Post on how to nest KVM under KVM.

Open Compute Project: The Open Compute Project Foundation's mission is to design and enable the delivery of the most efficient server, storage and data center hardware designs for scalable computing.

OpenStack Flavors: Official OpenStack documentation.

OpenStack High Availability Guide: Information on how to provide redundancy for the OpenStack components.

OpenStack Hypervisor Support Matrix: Matrix of supported hypervisors and capabilities when used with OpenStack.

OpenStack Object Store (Swift) Replication Reference: Developer documentation of Swift replication.

OpenStack Operations Guide: The OpenStack Operations Guide provides information on setting up and installing OpenStack.

OpenStack Security Guide: The OpenStack Security Guide provides information on securing OpenStack deployments.

OpenStack Training Marketplace: The OpenStack Market for training and Vendors providing training on OpenStack.

PCI passthrough: The PCI API patches extend the servers/os-hypervisor to show PCI information for instance and compute node, and also provides a resource endpoint to show PCI information.

TripleO: TripleO is a program aimed at installing, upgrading and operating OpenStack clouds using OpenStack's own cloud facilities as the foundation.

# Appendix A. Community support

## Table of Contents



The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

## Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- *Installation Guide for Debian 7.0*
- *Installation Guide for openSUSE and SUSE Linux Enterprise Server*
- *Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora*
- *Installation Guide for Ubuntu 14.04 (LTS)*

The following books explain how to configure and run an OpenStack cloud:

- *Architecture Design Guide*
- *Cloud Administrator Guide*
- *Configuration Reference*
- *Operations Guide*
- *High Availability Guide*
- *Security Guide*
- *Virtual Machine Image Guide*

The following books explain how to use the OpenStack dashboard and command-line clients:

- *API Quick Start*
- *End User Guide*
- *Admin User Guide*
- *Command-Line Interface Reference*

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)
- API Complete Reference (PDF)
- *OpenStack Block Storage Service API v2 Reference*
- *OpenStack Compute API v2 and Extensions Reference*
- *OpenStack Identity Service API v2.0 Reference*
- *OpenStack Image Service API v2 Reference*
- *OpenStack Networking API v2.0 Reference*
- *OpenStack Object Storage API v1 Reference*

The *Training Guides* offer software training for cloud administration and management.

# ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the http://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

# OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at http://wiki.openstack.org/MailingLists.

# The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

# The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, `"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- Bugs: OpenStack Block Storage (cinder)
- Bugs: OpenStack Compute (nova)
- Bugs: OpenStack Dashboard (horizon)
- Bugs: OpenStack Identity (keystone)
- Bugs: OpenStack Image Service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Bare Metal (ironic)
- Bugs: Data Processing Service (sahara)
- Bugs: Database Service (trove)
- Bugs: Orchestration (heat)
- Bugs: Telemetry (ceilometer)
- Bugs: Queue Service (marconi)
- Bugs: OpenStack API Documentation (developer.openstack.org)
- Bugs: OpenStack Documentation (docs.openstack.org)

# The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to http://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is `#openstack` on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

# Documentation feedback

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

# OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** http://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** http://openstack.redhat.com/
- **openSUSE and SUSE Linux Enterprise Server:** http://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive

# Glossary

6to4
: A mechanism that allows IPv6 packets to be transmitted over an IPv4 network, providing a strategy for migrating to IPv6.

Address Resolution Protocol (ARP)
: The protocol by which layer-3 IP addresses are resolved into layer-2 link local addresses.

Block Storage
: The OpenStack core project that enables management of volumes, volume snapshots, and volume types. The project name of Block Storage is cinder.

Border Gateway Protocol (BGP)
: The Border Gateway Protocol is a dynamic routing protocol that connects autonomous systems. Considered the backbone of the Internet, this protocol connects disparate networks to form a larger network.

bursting
: The practice of utilizing a secondary environment to elastically build instances on-demand when the primary environment is resource constrained.

ceilometer
: The project name for the Telemetry service, which is an integrated project that provides metering and measuring facilities for OpenStack.

cell
: Provides logical partitioning of Compute resources in a child and parent relationship. Requests are passed from parent cells to child cells if the parent cannot provide the requested resource.

cinder
: A core OpenStack project that provides block storage services for VMs.

Compute
: The OpenStack core project that provides compute services. The project name of Compute service is nova.

content delivery network (CDN)
: A content delivery network is a specialized network that is used to distribute content to clients, typically located close to the client for increased performance.

dashboard
: The web-based management interface for OpenStack. An alternative name for horizon.

Database Service
: An integrated project that provide scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines. The project name of Database Service is trove.

denial of service (DoS)
: Denial of service (DoS) is a short form for denial-of-service attack. This is a malicious attempt to prevent legitimate users from using a service.

Desktop-as-a-Service
: A platform that provides a suite of desktop environments that users may log in to receive a desktop experience from any location. This may provide general use, development, or even homogeneous testing environments.

east-west traffic
: Network traffic between servers in the same cloud or data center. See also north-south traffic.

encapsulation
: The practice of placing one packet type within another for the purposes of abstracting or securing data. Examples include GRE, MPLS, or IPsec.

glance
: A core project that provides the OpenStack Image Service.

heat
: An integrated project that aims to orchestrate multiple cloud applications for OpenStack.

Heat Orchestration Template (HOT)
: Heat input in the format native to OpenStack.

high availability (HA)
: A high availability system design approach and associated service implementation ensures that a prearranged level of operational performance will be met during a contractual measurement period. High availability systems seeks to minimize system downtime and data loss.

horizon
: OpenStack project that provides a dashboard, which is a web interface.

hybrid cloud
: A hybrid cloud is a composition of two or more clouds (private, community or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment models. Hybrid cloud can also mean the ability to connect colocation, managed and/or dedicated services with cloud resources.

IaaS
: Infrastructure-as-a-Service. IaaS is a provisioning model in which an organization outsources physical components of a data center, such as storage, hardware, servers, and networking components. A service provider owns the equipment and is responsible for housing, operating and maintaining it. The client typically pays on a per-use basis. IaaS is a model for providing cloud services.

Image Service
: An OpenStack core project that provides discovery, registration, and delivery services for disk and server images. The project name of the Image Service is glance.

IOPS
: IOPS (Input/Output Operations Per Second) are a common performance measurement used to benchmark computer storage devices like hard disk drives, solid state drives, and storage area networks.

kernel-based VM (KVM)
: An OpenStack-supported hypervisor. KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V), ARM, IBM Power, and IBM zSeries. It consists of a loadable kernel module, that provides the core virtualization infrastructure and a processor specific module.

keystone
: The project that provides OpenStack Identity services.

Layer-2 network
: Term used in the OSI network architecture for the data link layer. The data link layer is responsible for media access control, flow control and detecting and possibly correcting erros that may occur in the physical layer.

Layer-3 network
: Term used in the OSI network architecture for the network layer. The network layer is responsible for packet forwarding including routing from one node to another.

Networking
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute. The project name of Networking is neutron.

neutron
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute.

north-south traffic
: Network traffic between a user or client (north) and a server (south), or traffic into the cloud (south) and out of the cloud (north). See also east-west traffic.

nova
: OpenStack project that provides compute services.

Object Storage
: The OpenStack core project that provides eventually consistent and redundant storage and retrieval of fixed digital content. The project name of OpenStack Object Storage is swift.

Open vSwitch
: Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (for example NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).

OpenStack
: OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack is an open source project licensed under the Apache License 2.0.

Orchestration
: An integrated project that orchestrates multiple cloud applications for OpenStack. The project name of Orchestration is heat.

Platform-as-a-Service (PaaS)
: Provides to the consumer the ability to deploy applications through a programming language or tools supported by the cloud platform provider. An example of Platform-as-a-Service is an Eclipse/Java programming platform provided with no downloads required.

swift
: An OpenStack core project that provides object storage services.

Telemetry
: An integrated project that provides metering and measuring facilities for OpenStack. The project name of Telemetry is ceilometer.

TripleO
: OpenStack-on-OpenStack program. The code name for the OpenStack Deployment program.

trove
: OpenStack project that provides database services to applications.

Xen

Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.